

Go 1.13 Release Notes



- [Introduction to Go 1.13 {#introduction}](#)
- [Changes to the language {#language}](#)
- [Ports {#ports}](#)
 - [AIX {#aix}](#)
 - [Android {#android}](#)
 - [Darwin {#darwin}](#)
 - [FreeBSD {#freebsd}](#)
 - [Illumos {#illumos}](#)
 - [Windows {#windows}](#)
- [Tools {#tools}](#)
 - [Modules {#modules}](#)
 - [Environment variables {#proxy-vars}](#)
 - [go get {#go-get}](#)
 - [Version validation {#version-validation}](#)
 - [Go command {#go-command}](#)
 - [Compiler toolchain {#compiler}](#)
 - [Assembler {#assembler}](#)
 - [gofmt {#gofmt}](#)
 - [godoc and go doc {#godoc}](#)
- [Runtime {#runtime}](#)
- [Standard library {#library}](#)
 - [TLS 1.3 {#tls_1_3}](#)
 - [crypto/ed25519 {#crypto_ed25519}](#)
 - [Error wrapping {#error_wrapping}](#)
 - [Minor changes to the library {#minor_library_changes}](#)
 - [bytes](#)
 - [context](#)
 - [crypto/tls](#)

- [crypto/x509](#)
 - [database/sql](#)
 - [debug/dwarf](#)
 - [errors](#)
 - [fmt](#)
 - [go/scanner](#)
 - [go/types](#)
 - [html/template](#)
 - [log](#)
 - [math/big](#)
 - [math/bits](#)
 - [net](#)
 - [net/http](#)
 - [os](#)
 - [os/exec](#)
 - [reflect](#)
 - [runtime](#)
 - [strconv](#)
 - [strings](#)
 - [sync](#)
 - [syscall](#)
 - [syscall/js](#)
 - [testing](#)
 - [text/scanner](#)
 - [text/template](#)
 - [time](#)
 - [unicode](#)
-

Introduction to Go 1.13

{ #introduction }

The latest Go release, version 1.13, arrives six months after [Go 1.12](#). 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.

As of Go 1.13, the `go` command by default downloads and authenticates modules using the Go module mirror and Go checksum database run by Google. See <https://proxy.golang.org/privacy> for privacy information about these services and the [go command documentation](#) for configuration details including how to disable the use of these servers or use different ones. If you depend on non-public modules, see the [documentation for configuring your environment](#).

Changes to the language

{#language}

Per the [number literal proposal](#), Go 1.13 supports a more uniform and modernized set of number literal prefixes.

- [Binary integer literals](#): The prefix `0b` or `0B` indicates a binary integer literal such as `0b1011`.
- [Octal integer literals](#): The prefix `0o` or `0O` indicates an octal integer literal such as `0o660`. The existing octal notation indicated by a leading `0` followed by octal digits remains valid.
- [Hexadecimal floating point literals](#): The prefix `0x` or `0X` may now be used to express the mantissa of a floating-point number in hexadecimal format such as `0x1.0p-1021`. A hexadecimal floating-point number must always have an exponent, written as the letter `p` or `P` followed by an exponent in decimal. The exponent scales the mantissa by 2 to the power of the exponent.
- [Imaginary literals](#): The imaginary suffix `i` may now be used with any (binary, decimal, hexadecimal) integer or floating-point literal.
- Digit separators: The digits of any number literal may now be separated (grouped) using underscores, such as in `1_000_000`, `0b_1010_0110`, or `3.1415_9265`. An underscore may appear between any two digits or the literal prefix and the first digit.

Per the [signed shift counts proposal](#) Go 1.13 removes the restriction that a [shift count](#) must be unsigned. This change eliminates the need for many artificial `uint` conversions, solely introduced to satisfy this (now removed) restriction of the `<<` and `>>` operators.

These language changes were implemented by changes to the compiler, and corresponding internal changes to the library packages [go/scanner](#) and [text/scanner](#) (number literals), and

`go/types` (signed shift counts).

If your code uses modules and your `go.mod` file specifies a language version, be sure it is set to at least `1.13` to get access to these language changes. You can do this by editing the `go.mod` file directly, or you can run `go mod edit -go=1.13`.

Ports {#ports}

Go 1.13 is the last release that will run on Native Client (NaCl).

For `GOARCH=wasm`, the new environment variable `GOWASM` takes a comma-separated list of experimental features that the binary gets compiled with. The valid values are documented [here](#).

AIX {#aix}

AIX on PPC64 (`aix/ppc64`) now supports cgo, external linking, and the `c-archive` and `pie` build modes.

Android {#android}

Go programs are now compatible with Android 10.

Darwin {#darwin}

As [announced](#) in the Go 1.12 release notes, Go 1.13 now requires macOS 10.11 El Capitan or later; support for previous versions has been discontinued.

FreeBSD {#freebsd}

As [announced](#) in the Go 1.12 release notes, Go 1.13 now requires FreeBSD 11.2 or later; support for previous versions has been discontinued. FreeBSD 12.0 or later requires a kernel with the `COMPAT_FREEBSD11` option set (this is the default).

Illumos {#illumos}

Go now supports Illumos with `GOS=illumos`. The `illumos` build tag implies the `solaris` build tag.

Windows {#windows}

The Windows version specified by internally-linked Windows binaries is now Windows 7 rather than NT 4.0. This was already the minimum required version for Go, but can affect the behavior of system calls that have a backwards-compatibility mode. These will now behave as documented. Externally-linked binaries (any program using cgo) have always specified a more recent Windows version.

Tools {#tools}

Modules {#modules}

Environment variables {#proxy-vars}

The `GOL11MODULE` environment variable continues to default to `auto`, but the `auto` setting now activates the module-aware mode of the `go` command whenever the current working directory contains, or is below a directory containing, a `go.mod` file — even if the current directory is within `GOPATH/src`. This change simplifies the migration of existing code within `GOPATH/src` and the ongoing maintenance of module-aware packages alongside non-module-aware importers.

The new `GOPRIVATE` environment variable indicates module paths that are not publicly available. It serves as the default value for the lower-level `GONOPROXY` and `GONOSUMDB` variables, which provide finer-grained control over which modules are fetched via proxy and verified using the checksum database.

The `GOPROXY` environment variable may now be set to a comma-separated list of proxy URLs or the special token `direct`, and its default value is now `https://proxy.golang.org,direct`. When resolving a package path to its containing module, the `go` command will try all candidate module paths on each proxy in the list in succession. An unreachable proxy or HTTP status code other than 404 or 410 terminates the search without consulting the remaining proxies.

The new `GOSUMDB` environment variable identifies the name, and optionally the public key and server URL, of the database to consult for checksums of modules that are not yet listed in the main module's `go.sum` file. If `GOSUMDB` does not include an explicit URL, the URL is chosen by probing the `GOPROXY` URLs for an endpoint indicating support for the checksum database, falling back to a direct connection to the named database if it is not supported by any proxy. If `GOSUMDB` is set to `off`, the checksum database is not consulted and only the existing checksums in the `go.sum` file are verified.

Users who cannot reach the default proxy and checksum database (for example, due to a firewalled or sandboxed configuration) may disable their use by setting `GOPROXY` to `direct`, and/or

`GOSUMDB` to `off`. `go env -w` can be used to set the default values for these variables independent of platform:

```
go env -w GOPROXY=direct
go env -w GOSUMDB=off
```

`go get {#go-get}`

In module-aware mode, `go get` with the `-u` flag now updates a smaller set of modules that is more consistent with the set of packages updated by `go get -u` in GOPATH mode. `go get -u` continues to update the modules and packages named on the command line, but additionally updates only the modules containing the packages *imported by* the named packages, rather than the transitive module requirements of the modules containing the named packages.

Note in particular that `go get -u` (without additional arguments) now updates only the transitive imports of the package in the current directory. To instead update all of the packages transitively imported by the main module (including test dependencies), use `go get -u all`.

As a result of the above changes to `go get -u`, the `go get` subcommand no longer supports the `-m` flag, which caused `go get` to stop before loading packages. The `-d` flag remains supported, and continues to cause `go get` to stop after downloading the source code needed to build dependencies of the named packages.

By default, `go get -u` in module mode upgrades only non-test dependencies, as in GOPATH mode. It now also accepts the `-t` flag, which (as in GOPATH mode) causes `go get` to include the packages imported by *tests of* the packages named on the command line.

In module-aware mode, the `go get` subcommand now supports the version suffix `@patch`. The `@patch` suffix indicates that the named module, or module containing the named package, should be updated to the highest patch release with the same major and minor versions as the version found in the build list.

If a module passed as an argument to `go get` without a version suffix is already required at a newer version than the latest released version, it will remain at the newer version. This is consistent with the behavior of the `-u` flag for module dependencies. This prevents unexpected downgrades from pre-release versions. The new version suffix `@upgrade` explicitly requests this behavior. `@latest` explicitly requests the latest version regardless of the current version.

Version validation {#version-validation}

When extracting a module from a version control system, the `go` command now performs additional validation on the requested version string.

The `+incompatible` version annotation bypasses the requirement of [semantic import versioning](#) for repositories that predate the introduction of modules. The `go` command now verifies that such a

version does not include an explicit `go.mod` file.

The `go` command now verifies the mapping between [pseudo-versions](#) and version-control metadata. Specifically:

- The version prefix must be of the form `vX.0.0`, or derived from a tag on an ancestor of the named revision, or derived from a tag that includes [build metadata](#) on the named revision itself.
- The date string must match the UTC timestamp of the revision.
- The short name of the revision must use the same number of characters as what the `go` command would generate. (For SHA-1 hashes as used by `git`, a 12-digit prefix.)

If a `require` directive in the [main module](#) uses an invalid pseudo-version, it can usually be corrected by redacting the version to just the commit hash and re-running a `go` command, such as `go list -m all` or `go mod tidy`. For example,

```
require github.com/docker/docker v1.14.0-0.20190319215453-e7b5f7dbe98c
```

can be redacted to

```
require github.com/docker/docker e7b5f7dbe98c
```

which currently resolves to

```
require github.com/docker/docker v0.7.3-0.20190319215453-e7b5f7dbe98c
```

If one of the transitive dependencies of the main module requires an invalid version or pseudo-version, the invalid version can be replaced with a valid one using a [replace directive](#) in the `go.mod` file of the main module. If the replacement is a commit hash, it will be resolved to the appropriate pseudo-version as above. For example,

```
replace github.com/docker/docker v1.14.0-0.20190319215453-e7b5f7dbe98c =>
github.com/docker/docker e7b5f7dbe98c
```

currently resolves to

```
replace github.com/docker/docker v1.14.0-0.20190319215453-e7b5f7dbe98c =>
github.com/docker/docker v0.7.3-0.20190319215453-e7b5f7dbe98c
```

Go command {#go-command}

The `go env` command now accepts a `-w` flag to set the per-user default value of an environment variable recognized by the `go` command, and a corresponding `-u` flag to unset a previously-set default. Defaults set via `go env -w` are stored in the `go/env` file within `os.UserConfigDir()`.

The `go version` command now accepts arguments naming executables and directories. When invoked on an executable, `go version` prints the version of Go used to build the executable. If the `-m` flag is used, `go version` prints the executable's embedded module version information, if available. When invoked on a directory, `go version` prints information about executables contained in the directory and its subdirectories.

The new `go build flag -trimpath` removes all file system paths from the compiled executable, to improve build reproducibility.

If the `-o` flag passed to `go build` refers to an existing directory, `go build` will now write executable files within that directory for `main` packages matching its package arguments.

The `go build` flag `-tags` now takes a comma-separated list of build tags, to allow for multiple tags in `GOFLAGS`. The space-separated form is deprecated but still recognized and will be maintained.

`go generate` now sets the `generate` build tag so that files may be searched for directives but ignored during build.

As [announced](#) in the Go 1.12 release notes, binary-only packages are no longer supported. Building a binary-only package (marked with a `//go:binary-only-package` comment) now results in an error.

Compiler toolchain {#compiler}

The compiler has a new implementation of escape analysis that is more precise. For most Go code should be an improvement (in other words, more Go variables and expressions allocated on the stack instead of heap). However, this increased precision may also break invalid code that happened to work before (for example, code that violates the `unsafe.Pointer` [safety rules](#)). If you notice any regressions that appear related, the old escape analysis pass can be re-enabled with `go build -gcflags=all=-newescape=false`. The option to use the old escape analysis will be removed in a future release.

The compiler no longer emits floating point or complex constants to `go_asm.h` files. These have always been emitted in a form that could not be used as numeric constant in assembly code.

Assembler {#assembler}

The assembler now supports many of the atomic instructions introduced in ARM v8.1.

gofmt {#gofmt}

`gofmt` (and with that `go fmt`) now canonicalizes number literal prefixes and exponents to use lower-case letters, but leaves hexadecimal digits alone. This improves readability when using the new octal prefix (`00` becomes `0o`), and the rewrite is applied consistently. `gofmt` now also removes unnecessary leading zeroes from a decimal integer imaginary literal. (For backwards-compatibility, an integer imaginary literal starting with `0` is considered a decimal, not an octal number. Removing superfluous leading zeroes avoids potential confusion.) For instance, `0B1010`, `0XabcDEF`, `00660`, `1.2E3`, and `01i` become `0b1010`, `0xabcDEF`, `0o660`, `1.2e3`, and `1i` after applying `gofmt`.

godoc and go doc {#godoc}

The `godoc` webserver is no longer included in the main binary distribution. To run the `godoc` webserver locally, manually install it first:

```
go get golang.org/x/tools/cmd/godoc
godoc
```

The `go doc` command now always includes the package clause in its output, except for commands. This replaces the previous behavior where a heuristic was used, causing the package clause to be omitted under certain conditions.

Runtime {#runtime}

Out of range panic messages now include the index that was out of bounds and the length (or capacity) of the slice. For example, `s[3]` on a slice of length 1 will panic with "runtime error: index out of range [3] with length 1".

This release improves performance of most uses of `defer` by 30%.

The runtime is now more aggressive at returning memory to the operating system to make it available to co-tenant applications. Previously, the runtime could retain memory for five or more minutes following a spike in the heap size. It will now begin returning it promptly after the heap shrinks. However, on many OSes, including Linux, the OS itself reclaims memory lazily, so process RSS will not decrease until the system is under memory pressure.

Standard library {#library}

TLS 1.3 {#tls_1_3}

As announced in Go 1.12, Go 1.13 enables support for TLS 1.3 in the `crypto/tls` package by default. It can be disabled by adding the value `tls13=0` to the `GODEBUG` environment variable. The opt-out will be removed in Go 1.14.

See [the Go 1.12 release notes](#) for important compatibility information.

[crypto/ed25519](#) {#crypto_ed25519}

The new `crypto/ed25519` package implements the Ed25519 signature scheme. This functionality was previously provided by the [golang.org/x/crypto/ed25519](#) package, which becomes a wrapper for `crypto/ed25519` when used with Go 1.13+.

Error wrapping {#error_wrapping}

Go 1.13 contains support for error wrapping, as first proposed in the [Error Values proposal](#) and discussed on [the associated issue](#).

An error `e` can *wrap* another error `w` by providing an `Unwrap` method that returns `w`. Both `e` and `w` are available to programs, allowing `e` to provide additional context to `w` or to reinterpret it while still allowing programs to make decisions based on `w`.

To support wrapping, `fmt.Errorf` now has a `%w` verb for creating wrapped errors, and three new functions in the `errors` package (`errors.Unwrap`, `errors.Is` and `errors.As`) simplify unwrapping and inspecting wrapped errors.

For more information, read the [errors package documentation](#), or see the [Error Value FAQ](#). There will soon be a blog post as well.

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.

[bytes](#)

The new `ToValidUTF8` function returns a copy of a given byte slice with each run of invalid UTF-8 byte sequences replaced by a given slice.

context

The formatting of contexts returned by `WithValue` no longer depends on `fmt` and will not stringify in the same way. Code that depends on the exact previous stringification might be affected.

crypto/tls

Support for SSL version 3.0 (SSLv3) [is now deprecated and will be removed in Go 1.14](#). Note that SSLv3 is the [cryptographically broken](#) protocol predating TLS.

SSLv3 was always disabled by default, other than in Go 1.12, when it was mistakenly enabled by default server-side. It is now again disabled by default. (SSLv3 was never supported client-side.)

Ed25519 certificates are now supported in TLS versions 1.2 and 1.3.

crypto/x509

Ed25519 keys are now supported in certificates and certificate requests according to [RFC 8410](#), as well as by the `ParsePKCS8PrivateKey`, `MarshalPKCS8PrivateKey`, and `ParsePKIXPublicKey` functions.

The paths searched for system roots now include `/etc/ssl/cert.pem` to support the default location in Alpine Linux 3.7+.

database/sql

The new `NullTime` type represents a `time.Time` that may be null.

The new `NullInt32` type represents an `int32` that may be null.

debug/dwarf

The `Data.Type` method no longer panics if it encounters an unknown DWARF tag in the type graph. Instead, it represents that component of the type with an `UnsupportedType` object.

errors

The new function `As` finds the first error in a given error's chain (sequence of wrapped errors) that matches a given target's type, and if so, sets the target to that error value.

The new function `Is` reports whether a given error value matches an error in another's chain.

The new function `Unwrap` returns the result of calling `Unwrap` on a given error, if one exists.

fmt

The printing verbs `%x` and `%X` now format floating-point and complex numbers in hexadecimal notation, in lower-case and upper-case respectively.

The new printing verb `%0` formats integers in base 8, emitting the `0o` prefix.

The scanner now accepts hexadecimal floating-point values, digit-separating underscores and leading `0b` and `0o` prefixes. See the [Changes to the language](#) for details.

The `Errorf` function has a new verb, `%w`, whose operand must be an error. The error returned from `Errorf` will have an `Unwrap` method which returns the operand of `%w`.

go/scanner

The scanner has been updated to recognize the new Go number literals, specifically binary literals with `0b`/`0B` prefix, octal literals with `0o`/`0O` prefix, and floating-point numbers with hexadecimal mantissa. The imaginary suffix `i` may now be used with any number literal, and underscores may be used as digit separators for grouping. See the [Changes to the language](#) for details.

go/types

The type-checker has been updated to follow the new rules for integer shifts. See the [Changes to the language](#) for details.

html/template

When using a `<script>` tag with "module" set as the type attribute, code will now be interpreted as [JavaScript module script](#).

log

The new `Writer` function returns the output destination for the standard logger.

math/big

The new `Rat.SetUint64` method sets the `Rat` to a `uint64` value.

For `Float.Parse`, if base is 0, underscores may be used between digits for readability. See the [Changes to the language](#) for details.

For `Int.SetString`, if base is 0, underscores may be used between digits for readability. See the [Changes to the language](#) for details.

`Rat.SetString` now accepts non-decimal floating point representations.

[math/bits](#)

The execution time of `Add`, `Sub`, `Mul`, `RotateLeft`, and `ReverseBytes` is now guaranteed to be independent of the inputs.

[net](#)

On Unix systems where `use-vc` is set in `resolv.conf`, TCP is used for DNS resolution.

The new field `ListenConfig.KeepAlive` specifies the keep-alive period for network connections accepted by the listener. If this field is 0 (the default) TCP keep-alives will be enabled. To disable them, set it to a negative value.

Note that the error returned from I/O on a connection that was closed by a keep-alive timeout will have a `Timeout` method that returns `true` if called. This can make a keep-alive error difficult to distinguish from an error returned due to a missed deadline as set by the `SetDeadline` method and similar methods. Code that uses deadlines and checks for them with the `Timeout` method or with `os.IsTimeout` may want to disable keep-alives, or use `errors.Is(syscall.ETIMEDOUT)` (on Unix systems) which will return true for a keep-alive timeout and false for a deadline timeout.

[net/http](#)

The new fields `Transport.WriteBufferSize` and `Transport.ReadBufferSize` allow one to specify the sizes of the write and read buffers for a `Transport`. If either field is zero, a default size of 4KB is used.

The new field `Transport.ForceAttemptHTTP2` controls whether HTTP/2 is enabled when a non-zero `Dial`, `DialTLS`, or `DialContext` func or `TLSCClientConfig` is provided.

`Transport.MaxConnsPerHost` now works properly with HTTP/2.

`TimeoutHandler`'s `ResponseWriter` now implements the `Pusher` interface.

The `StatusCode` 103 "Early Hints" has been added.

`Transport` now uses the `Request.Body`'s `io.ReaderFrom` implementation if available, to optimize writing the body.

On encountering unsupported transfer-encodings, `http.Server` now returns a "501 Unimplemented" status as mandated by the HTTP specification [RFC 7230 Section 3.3.1](#).

The new `Server` fields `BaseContext` and `ConnContext` allow finer control over the `Context` values provided to requests and connections.

`http.DetectContentType` now correctly detects RAR signatures, and can now also detect RAR v5 signatures.

The new `Header` method `Clone` returns a copy of the receiver.

A new function `NewRequestWithContext` has been added and it accepts a `Context` that controls the entire lifetime of the created outgoing `Request`, suitable for use with `Client.Do` and `Transport.RoundTrip`.

The `Transport` no longer logs errors when servers gracefully shut down idle connections using a "408 Request Timeout" response.

os

The new `UserConfigDir` function returns the default directory to use for user-specific configuration data.

If a `File` is opened using the `O_APPEND` flag, its `WriteAt` method will always return an error.

os/exec

On Windows, the environment for a `Cmd` always inherits the `%SYSTEMROOT%` value of the parent process unless the `Cmd.Env` field includes an explicit value for it.

reflect

The new `Value.IsZero` method reports whether a `Value` is the zero value for its type.

The `MakeFunc` function now allows assignment conversions on returned values, instead of requiring exact type match. This is particularly useful when the type being returned is an interface type, but the value actually returned is a concrete value implementing that type.

runtime

Tracebacks, `runtime.Caller`, and `runtime.Callers` now refer to the function that initializes the global variables of `PKG` as `PKG.init` instead of `PKG.initializers`.

strconv

For `strconv.ParseFloat`, `strconv.ParseInt` and `strconv.ParseUint`, if base is 0, underscores may be used between digits for readability. See the [Changes to the language](#) for details.

strings

The new `ToValidUTF8` function returns a copy of a given string with each run of invalid UTF-8 byte sequences replaced by a given string.

sync

The fast paths of `Mutex.Lock`, `Mutex.Unlock`, `RWMutex.Lock`, `RWMutex.RUnlock`, and `Once.Do` are now inlined in their callers. For the uncontended cases on amd64, these changes make `Once.Do` twice as fast, and the `Mutex`/`RWMutex` methods up to 10% faster.

Large `Pool` no longer increase stop-the-world pause times.

`Pool` no longer needs to be completely repopulated after every GC. It now retains some objects across GCs, as opposed to releasing all objects, reducing load spikes for heavy users of `Pool`.

syscall

Uses of `_getdirentries64` have been removed from Darwin builds, to allow Go binaries to be uploaded to the macOS App Store.

The new `ProcessAttributes` and `ThreadAttributes` fields in `SysProcAttr` have been introduced for Windows, exposing security settings when creating new processes.

`EINVAL` is no longer returned in zero `Chmod` mode on Windows.

Values of type `Errno` can be tested against error values in the `os` package, like `ErrExist`, using `errors.Is`.

syscall/js

`TypedArrayOf` has been replaced by `CopyBytesToGo` and `CopyBytesToJS` for copying bytes between a byte slice and a `Uint8Array`.

testing

When running benchmarks, `B.N` is no longer rounded.

The new method `B.ReportMetric` lets users report custom benchmark metrics and override built-in metrics.

Testing flags are now registered in the new `Init` function, which is invoked by the generated `main` function for the test. As a result, testing flags are now only registered when running a test binary, and packages that call `flag.Parse` during package initialization may cause tests to fail.

text/scanner

The scanner has been updated to recognize the new Go number literals, specifically binary literals with `0b`/`0B` prefix, octal literals with `0o`/`0O` prefix, and floating-point numbers with hexadecimal mantissa. Also, the new `AllowDigitSeparators` mode allows number literals to contain underscores as digit separators (off by default for backwards-compatibility). See the [Changes to the language](#) for details.

text/template

The new [slice function](#) returns the result of slicing its first argument by the following arguments.

time

Day-of-year is now supported by `Format` and `Parse`.

The new `Duration` methods `Microseconds` and `Milliseconds` return the duration as an integer count of their respectively named units.

unicode

The `unicode` package and associated support throughout the system has been upgraded from Unicode 10.0 to [Unicode 11.0](#), which adds 684 new characters, including seven new scripts, and 66 new emoji.

```
00000000 #2
00000000 2026-10-27 04:26:29 UTC
00000000 2026-10-27 04:26:29 UTC
```