Gopher Academy Blog

Community Contributed Go Articles and Tutorials

Go 1.10

December 30, 2017
Contributed by Florin Pățan

Table of contents

Introduction

Go 1.10 is the first major release after the announcement of the plans towards Go 2.0 (https://blog.golang.org/toward-go2) at GopherCon 2017 (https://www.youtube.com/watch?v=0Zbh_vmAKvk).

There are a number of exciting changes which I'll cover below as well as some changes in the behavior of either tools or Go APIs which might result in an unexpected behavior compared to the previous change. I chose to flag these changes as "breaking change" in order to make it easier to identify them.

I've also tried to flag the CL that brought in the change as the discussions on them as well as the related changes in other CLs are a great source from learning how Go is organized, created, how features are reviewed, and hopefully inspire you to contribute (https://golang.org/doc/contribute.html) to the language itself by either participating in reviews or issues or fixing issues (like Needs Investigation (https://github.com/golang/go/labels/NeedsInvestigation) or Help Wanted (https://github.com/golang/go/labels/help%20wanted)).

Language changes

Let's start with the language changes that Go 1.10 brings. There are only a couple of rather small changes, none of them significant.

First, you'll be able to use an untyped constant as the index of an expression `x[1.0 << s]` where `s` is the untyped constant CL 60230 (https://golang.org/cl/60230)

The second change is that you can now use method expressions like this `struct{io.Reader}.Read`, even if this is a rather unusual way to do so CL 73233 (https://golang.org/cl/73233)

Operating systems support

Moving on to the operating system support, Go 1.10 will be the last Go version to run on OS X 10.8 Mountain Lion, OS X 10.9 Mavericks, or on OpenBSD 6.0.

FreeBSD 10.3 is now required to run Go, up from FreeBSD 9.3 CL 64910 (https://golang.org/cl/64910)

NetBSD is once again supported, but only in the unreleased version 8 and on `386` and `amd64`. `arm` support for NetBSD 8 was still broken at the time of writing, see Issue 23073 (https://golang.org/issue/23073)

On 32-bit MIPS systems you can now choose if you want emulation for floating point instructions or not via a new environment variable settings `GOMIPS=hardfloat` (the default) and `GOMIPS=softfloat` CL 37954 (https://golang.org/cl/37954)

Tooling

The bigger changes in Go 1.10 come from the tooling improvements it brings. They dramatically improve the quality of life for testing in large and very large projects and pave the road towards Go 2.0.

Environment variables

Since Go 1.9, `GOROOT` is inferred from the location of the `go` tool binary by default. However, if your application relied on that value at runtime via `runtime.GOROOT()` there was a bug which prevented it have the correct location. As of Go 1.10+ this bug was fixed and you can now use it as expected CL 61310 (https://golang.org/cl/61310)

A couple of new environment variables were added to the `go` command. `GOTMPDIR` will allow you to configure where the temporary files created by Go during compilation of your applications are stored. The default path is the same as in the previous Go versions, the operating system's temporary files directory. The other new environment variable is `GOCACHE` allows you to control where the `go` command will store the cached information that's reused in future builds CL 75475 (https://golang.org/cl/75475)

go build

`go build` can now detect changes in files on a source code level rather than rely on timestamps, which means that it will be more accurate in rebuilding the packages that have changed. In turn, this means that you will now be able to drop the usage of `-a` flag, which was previously used to force Go to rebuild packages in certain conditions.

Changes are coming to the `-asmflags`, `-gcflags`, `-gccgoflags`, and `-ldflags` flags. They will not be applied automatically to the list of all packages as before but only to the direct package that's being specified in the `build` command. You can still achieve the same functionality as before using the new special syntax for

these flags, `-ldflags=pattern=flags` such as:
`go build -ldflags=cmd/gofmt=-X=main.version=1.2.3 cmd/...`. This command allows you to build all the `/cmd/...` packages but it will apply the `-ldflags=-X=main.version=1.2.3` flag only to the `cmd/gofmt` package CL 76551 (https://golang.org/cl/76551)

To further speed up the builds, `go build -i` will not be necessary since now the build tool will have its own cache for build steps that do not do the install step, such as `go install` or `go get`. This means that you'll be able to switch between branches or experiment a lot more with the code without having to invoke `go install` or `go build -i` but just `go build`.

Are you a Windows user? Now you can use c-shared as a target for your libraries, thanks to CL 69091 (https://golang.org/cl/69091).

go install

`go install` also received some changes. Now it will install only the explicitly mentioned packages but not their dependencies. To restore the previous behaviour, you'll need to use this command as `go install -i` CL 75850 (https://golang.org/cl/75850)

This change as well as upcoming changes is significant if your tools depend on the packages to be installed in `$GOPATH/pkg` and always be fresh, with additional changes being required in order to restore the old behaviour.

go test

Speaking of caching, `go test` has seen a lot of changes as well. One of the most important is that `go test` will now cache the results of the tests if they meet certain criteria such as:

- the test executable and command line match a previous run
- the files and environment variables used by that run have not changed
- the results are successful
- the `go test` command received a list of packages to test, `go test ./...` for example
- the test command line uses a subset of the test flags, `-cpu`, `-list`, `-parallel`, `-run`, `-short`, and `-v`

When the above conditions are met, the first run will produce the output as expected then subsequent runs will simply reuse that output. The run time of the tests will also notify that the cached output is used, by displaying `(cached)` instead of the original test run time.

You can always force the tests to run by specifying the flag `-count=1`, this being considered the idiomatic way to handle this requirement. As a recap, the `-count` flag allows you to specify how many times a test or a benchmark runs.

This change is covered in CL 75631 (https://golang.org/cl/75631).

The second important change to `go test` is that now a subset of `go vet` checks will run before the tests run in order to detect issues with your code. These checks will be treated as build failures in case any of them will produce any result. Only very high accuracy checks are included in this step. To disable this new behavior you'll need to provide the `-vet=off` flag to the `go test` command CL 74356 (https://golang.org/cl/74356)

The coverage profile of tests can now be created when running the tests against multiple packages, which was a highly requested feature. Combined with the new way to use the `-coverpkg` flag, it means you'll be able to get the coverage for all the packages tested packages as well as their dependencies when multiple packages are

being tested by running `go test -coverpkg=all -coverprofile cover.out ./...` CL 76875 (https://golang.org/cl/76875) and CL 76876 (https://golang.org/cl/76876)

Test binaries will now always write to `stdout` when invoked via `go test` whereas before `stderr` could have been used sometimes CL 76871 (https://golang.org/cl/76871)

Tests running in parallel will now be better delimited by having `PAUSE` and `CONT` as status update lines when running with `-v` flag. This change allows tooling to better interpret the start and stop of parallel tests. The `-failfast` flag now will stop testing immediately on the first failure, with the caveat that parallel tests are still allowed to continue until they finish CL 74450 (https://golang.org/cl/74450)

Finally, `go test -json` will now output the format of the tests in json so that tooling such as IDEs can better present the results of the test. There is also a new command, `go tool test2json` that will produce convert the test output to json CL 76873 (https://golang.org/cl/76873) and CL 76872 (https://golang.org/cl/76872)

gofmt

Another set of tooling changes in Go 1.10 come from `gofmt` as it received a few updates. First, three index slice expressions containing complex expressions are now always formatted as `slice[start+1 : stop : capacity]` CL 67633 (https://golang.org/cl/67633)

The second change is that single-method interface literals written on a single line, which are sometimes used in type assertions, are no longer split onto multiple lines CL 66130 (https://golang.org/cl/66130)

The third one is that if a composite literal would include a comment and only comments, then the comment(s) will now be indented CL 74232 (https://golang.org/cl/74232)

If you use `gofmt` in your CI environment, you will see some failures because of these changes. The official position is that `gofmt` is not covered by the same set of compatibility promises as Go 1 itself, so these are not “breaking changes” but it’s rather a constant evolving specification, which can suffer changes on each new Go release. The recommendation is not to have `gofmt` enforced in the CI or have everyone use the same binary version for the application that formats your source code as well as checks it in the CI system.

A good news is that now all flags supported by `gofmt` are supported by `go fmt` as well.

go fix

`go fix` now replaces imports from `golang.org/x/net/context` with `context` which will help you migrate your code to a Go 1.9+ compatible code by running `go tool fix -r context your/package` CL 58590 (https://golang.org/cl/58590)

pprof

Go 1.10 will also bring an update to the `pprof` tool. This brings a host of improvements, among which an updated UI featuring a flame graph representation of the profiling data CL 75870 (https://golang.org/cl/75870)

Runtime

Go’s runtime received a few updates, with the first one I’ll cover being done on how the `LockOSThread` and `UnlockOSThread` mechanism works. If before, in nested calls, `UnlockOSThread` would need to be called only once to unlock the thread, now it will need to be called as many times as `LockOSThread` was called CL 45752

(https://golang.org/cl/45752)

You may have noticed the `<autogenerated>` frame (line) in the stack traces before. This is now hidden, unless a panic or other issue happens in it. This also means that if your code would call `runtime.Caller` with a certain number of skip frames, then this change will be a “breaking change” in its behaviour as the `<autogenerated>` frames will not be counted there either CL 45412 (https://golang.org/cl/45412)

Another important change in the Go runtime is the introduction of soft and hard goals (limits) for garbage collection CL 59970 (https://golang.org/cl/59970)

The soft limit is the current value of the GOGC while the hard limit is 10% higher than the soft limit. Heavy GC reliant applications (so far only benchmarks) shows that there’s an increase in the heap size of the application.

CGO support

CGO support has also received updates, with C typedefs such as `typedef X Y` now which means you’ll be able to use `C.X` and `C.Y` interchangeably in Go now, as if they would be Go aliases, `type X = Y` CL 62670 (https://golang.org/cl/62670)

Another welcomed change when working with C and Go is that you can now pass Go strings directly to C. This is done by declaring a C function in a Go file with a parameter type of the special type name `_GoString_`. To access the string length, you’ll need to call `size_t _GoStringLen(_GoString_ s)` an in order to get the pointer to the string contents, you’ll need use `const char *_GoStringPtr(_GoString_ s)` CL 70890 (https://golang.org/cl/70890)

Some C types that were previously mapped to a pointer type in Go are now mapped to `uintptr` type. A couple of these types are `CFTypeRef` in Darwin’s CoreFoundation framework and the `jobject` in Java’s JNI interface. You’ll need to initialize the values for the affected types with `0` instead of `nil` This is a breaking change but thankfully you can fix this quickly and automatically by running `go tool fix -r cftype your/package` or `go tool fix -r jni your/package` CL 66332 (https://golang.org/cl/66332) and CL 81876 (https://golang.org/cl/81876)

Debugging

Debugging support has also been improved in the latest release which should make your debugging experience via Delve (https://github.com/derekparker/delve) even better than before. And as a reminder, Delve is most likely integrated with your favorite code editor.

Assembly support

Assembly support got better as well, with a host of new instructions being added. Most important changes are under amd64 platform with 359 new instructions including the full AVX, AVX2, BMI, BMI2, F16C, FMA3, SSE2, SSE3, SSSE3, SSE4.1, and SSE4.2 extension sets.

Packages

Changes in the various standard library packages:

- bufio (https://tip.golang.org/pkg/bufio) - the new Reader.Size (https://tip.golang.org/pkg/bufio/#Reader.Size) and Writer.Size (https://tip.golang.org/pkg/bufio/#Writer.Size) and methods report the Reader (https://tip.golang.org/pkg/bufio/#Reader) or Writer (https://tip.golang.org/pkg/bufio/#Writer)'s underlying buffer size CL 75150 (https://golang.org/cl/75150)
- bytes (https://tip.golang.org/pkg/bytes/) - the Fields (https://tip.golang.org/pkg/bytes/#Fields), FieldsFunc (https://tip.golang.org/pkg/bytes/#FieldsFunc), Split (https://tip.golang.org/pkg/bytes/#Split), and SplitAfter (https://tip.golang.org/pkg/bytes/#SplitAfter) each already returned slices pointing into the same underlying array as its input. Go 1.10 changes each of the returned subslices to have capacity equal to its length, so that appending to a subslice will not overwrite adjacent data in the original input. This is also a "breaking change" in the behavior of these functions and you might need to update your code.
- crypto/tls (https://tip.golang.org/pkg/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 CL 74950 (https://golang.org/cl/74950)
- crypto/x509 (https://tip.golang.org/pkg/crypto/x509/) - leaf certificate validation 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 CL 62693 (https://golang.org/cl/62693)
- database/sql/driver (https://tip.golang.org/pkg/database/sql/driver/) - 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
- encoding/json (https://tip.golang.org/pkg/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. CL 27231 (https://golang.org/cl/27231) 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. This means you may need to update your code or a "breaking change" will happen, which could be hidden if the code is not properly handling errors CL 76851 (https://golang.org/cl/76851)
- text/template (https://tip.golang.org/pkg/text/template/) and html/template (https://tip.golang.org/pkg/html/template/) - the new actions `{{break}}` and `{{continue}}` break out of the innermost `{{range ...}}` loop, like the corresponding Go statements CL 66410 (https://golang.org/cl/66410)
- math/rand (https://tip.golang.org/pkg/math/rand/) - the new math/rand.Shuffle (https://tip.golang.org/pkg/math/rand/#Shuffle) function and corresponding math/rand.*Rand.Shuffle (https://tip.golang.org/pkg/math/rand/#Rand.Shuffle) method shuffle an input sequence CL 51891 (https://golang.org/cl/51891)
- math (https://tip.golang.org/pkg/math/) - the new functions Round (https://tip.golang.org/pkg/math/#Round) and RoundToEven (https://tip.golang.org/pkg/math/#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 CL 43652 (https://golang.org/cl/43652) and CL 61211 (https://golang.org/cl/61211)

- net (https://tip.golang.org/pkg/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 (https://tip.golang.org/pkg/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/url (https://tip.golang.org/pkg/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
- os (https://tip.golang.org/pkg/os/) - `File` adds new methods SetDeadline (https://tip.golang.org/pkg/os/#File.SetDeadline), SetReadDeadline (https://tip.golang.org/pkg/os/#File.SetReadDeadline), and SetWriteDeadline (https://tip.golang.org/pkg/os/#File.SetWriteDeadline) that allow setting I/O deadlines when the underlying file descriptor supports non-blocking I/O operations CL 71770 (https://golang.org/cl/71770) The definition of these methods matches those in `net.Conn`. Also matching `net.Conn`, `File`'s `Close` method now guarantee that when `Close` returns, the underlying file descriptor has been closed
- strings (https://tip.golang.org/pkg/strings/) - a new type Builder (https://tip.golang.org/pkg/strings/#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 CL 74931 (https://golang.org/cl/74931)
- unicode (https://tip.golang.org/pkg/unicode/) - the `unicode` package and associated support throughout the system has been upgraded from version 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

A lot more packages have received changes but I've tried to keep the list to a minimum. To view the full list of changes, you can read this the Draft Release Notes (https://tip.golang.org/doc/go1.10).

Closing notes

Due to the vast amount of changes both in compiler and in runtime, some workloads are expected to perform better as of Go 1.10. However, I highly recommend that you grab the latest Go 1.10 release, 1.10 Beta 1 (https://golang.org/dl/#unstable) and test it on your workloads, run the tests against the new Go version and

help the Go team identify issues before Go 1.10 lands in February. There are a few weeks where even just running the test / benchmark suite could make the difference. And Go 1.10 Beta 1 is also available as a Docker container (https://hub.docker.com/_/golang/) so you can minimize the impact it has on your system.

If you want to stay up to date with the developments of Go, I recommend following @golang_cls (https://twitter.com/golang_cls) Twitter account which provides a curated list of interesting commits as they are added to Go.

If you want to talk more about Go, its evolution, and what's next for Go, tweet me (https://twitter.com/dlsniper), or meet me at Gophercon Iceland (https://gophercon.is)!

Finally, I would like to thank the Go Team and all contributors that helped Go reach 1.10 and I look forward to what the future holds.

A big thank you goes to Russ Cox and the team that created the initial draft documentation which this article uses / reuses a lot.

Errata:

The initial version of this article incorrectly mentioned a change about how GOROOT is handled. Thank you to Dominik Honnef (https://twitter.com/dominikhonnef) for reporting this.

1 comment • a year ago

Frank — The Advent series is awesome, thanks to all the authors and gopher academy (and gryski). Looks like there's [<65;52;17M>

Gleam: Distributed Map Reduce for Golang

11 comments • a year ago

Matthew Nguyen — Chris, my email is mattdinhnguyen@gmail.com. Thanks.

3 comments • a month ago

Marko Mudrinić — Template naming is mostly useful when working with [the nested templates]

Seeking around in an HTTP object

1 comment • 2 months ago

roger peppe — "maybe one of you readers would like to show us how to speed up Go's tar using a replacement for bufio next advent

Registrations

General Admission - Corporate

0

Ends Jul 16

General Admission Ticket - My company is paying the registration fee.

Read more

Price	US$895
TOTAL	US$895

General Admission - Self Paid

0

Ends Jul 16

General Admission Ticket - I am paying my own way.

Read more

Price	US$695
TOTAL	US$695

Order Now

Search The Blog

TAGS:

SERIES LIST

Advent 2013 (/series/advent-2013)
Advent 2014 (/series/advent-2014)
Advent 2015 (/series/advent-2015)
Advent 2016 (/series/advent-2016)

Advent 2017 (/series/advent-2017)
Birthday Bash 2014 (/series/birthday-bash-2014)
GopherCon 2015 (/series/gophercon-2015)
GopherCon 2016 (/series/gophercon-2016)
GopherCon 2017 (/series/gophercon-2017)

info@gopheracademy.com, +1 650 386 0795