Go 1.10 Release Party

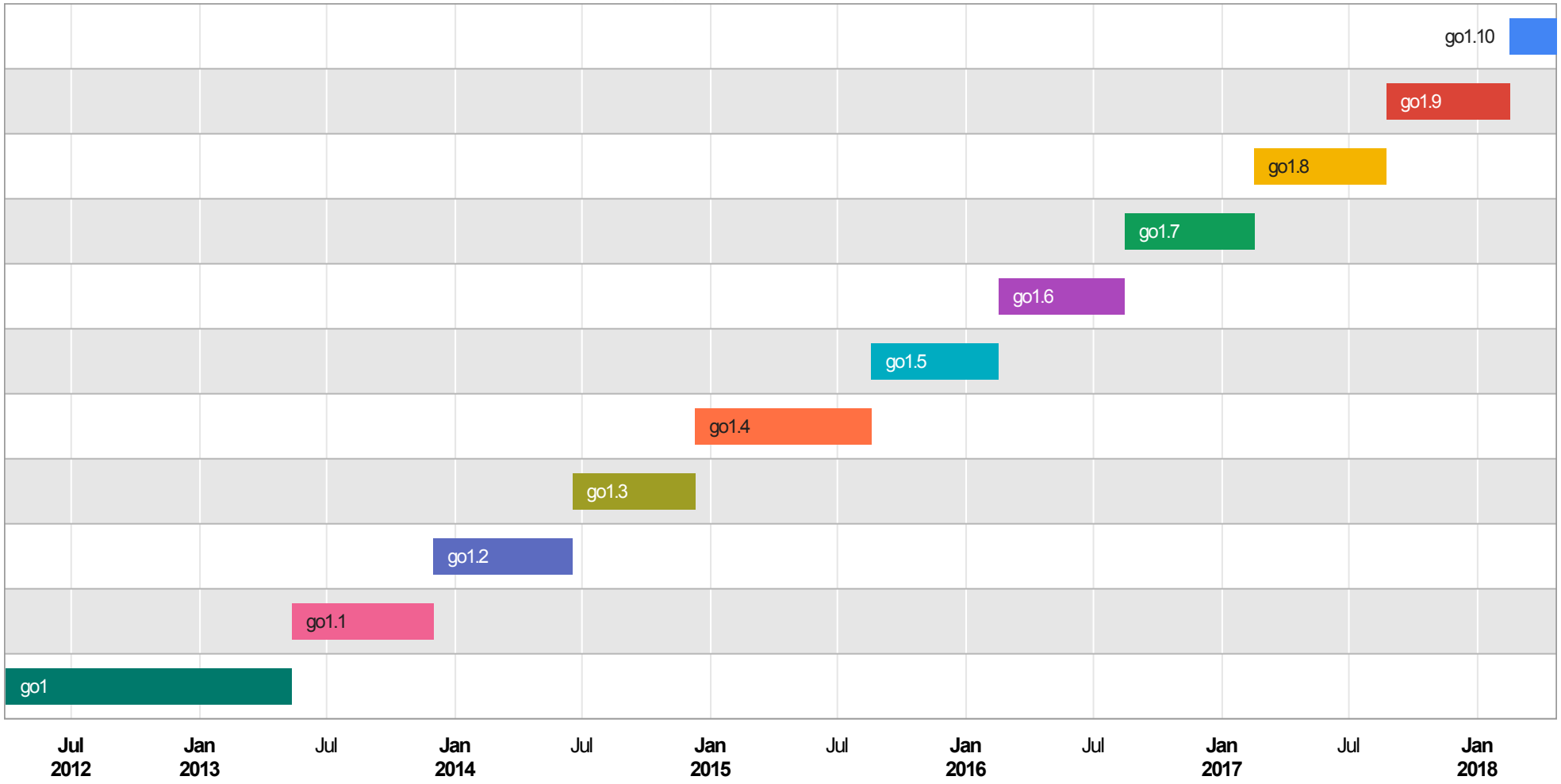
PDX Go
24 April 2018

Rodolfo Carvalho
AWS Elemental

Go

Go Releases

golang.org/doc/devel/release.html



Go 1 promise

golang.org/doc/go1compat

> Go 1 defines two things: first, the **specification of the language**; and second, the specification of a set of core APIs, the "**standard packages**" of the Go library.
>
> It is intended that programs written to the Go 1 specification will continue to compile and run correctly, unchanged, over the lifetime of that specification.
>
> Compatibility is at the source level.

Go 1 promise

> There are a number of ways in which a program that compiles and runs today may fail to do so after a future point release:
>
> - Security
> - Unspecified behavior
> - Specification errors
> - Bugs
> - Unkeyed struct literals
> - Methods creating conflict
> - Dot imports creating conflict
> - Use of package `unsafe`

Go 1 promise

> Although these expectations apply to Go 1 itself, we hope similar considerations would be made for the development of externally developed software based on Go 1.

Go 1.10

Release Notes in golang.org/doc/go1.10 fit in 14 letter pages!

- Language Specification changes
- go tool improvements
- Standard Library
- Ports
- Runtime
- Cgo improvements
- Performance

❤️ Hope you will learn at least one new thing before the end of the talk ❤️

Spec changes

Spec changes

Only minor changes to the language:

- shifts of untyped constants
- relaxed method expressions grammar

Untyped constants

Untyped constants

blog.golang.org/constants

> When designing Go, we decided to avoid this minefield [of numeric bugs in C code] by mandating that there is **no** mixing of numeric types.
>
> This strictness [...] sometimes requires programmers to decorate their code with clumsy numeric conversions to express their meaning clearly.
>
> [...] numeric constants work differently from how they behave in other C-like languages.
>
> In short, constants in Go just work, most of the time anyway.

Untyped constants

> The concept of *untyped constants* in Go means that all the numeric constants, whether integer, floating-point, complex, or even character values, live in a kind of unified space.
>
> It's when we bring them to the computational world of **variables**, **assignments**, and **operations** that the actual types matter.

Untyped constants

golang.org/ref/spec#Constants

Default type determined by syntax; actual type defined by context:

```
w := tabwriter.NewWriter(os.Stdout, 0, 0, 1, ' ', 0)
defer w.Flush()
describe := func(v interface{}) { fmt.Fprintf(w, "%T\t%[1]v\t\n", v) }

describe("Go 🤘")
describe(true)
describe(0)
describe(0.0)
describe('x')
describe(0i)

const c = 0.23 + 1.77
var x int64 = c
var y float32 = c
duration := c * time.Second
describe(x)
describe(y)
describe(duration)
```

Run

Shifting untyped constants

If the left operand of a non-constant shift expression is an untyped constant, it is **first converted** to the **type** it would **assume** if the shift expression were replaced by its left operand alone.

```
    a := []string{"Shifting", "untyped", "constant", "1.0", "converts", "to", "int"}
    var s uint = 2

    fmt.Println(a[1.0<<s]) // 1.0 has type int, 1.0<<s is int == 4

    // Output (go1.9 and earlier):
    // invalid operation: 1 << s (shift of type float64)

    // Output (go1.10):
    // converts
```

Run

```
var a = make([]byte, 1.0<<s) // 1.0 has type int
```

Method expressions

Method expressions

- golang.org/ref/spec#Method_expressions
- golang.org/ref/spec#Method_sets

```
type T struct {
    a int
}

func (tv T) Mv(a int) int           { return 0 } // value receiver
func (tp *T) Mp(f float32) float32 { return 1 } // pointer receiver

var t T
```

```
    for _, method := range []interface{}{
        T.Mv,
        (*T).Mp,
        (*T).Mv,
    } {
        fmt.Printf("%T\n", method)
    }
    fmt.Println(T.Mv(t, 42))
```

Run

Method expressions: relaxed grammar

#9060 spec: remove unnecessary syntax restrictions for method expressions to match compiler

```
    interface{ Mv(int) int }.Mv(t, 42) // valid
    struct{ T }.Mv(struct{ T }{}, 42)  // valid, but...
```

#22444 cmd/compile: missing wrapper function for call of literal method expression

```
# command-line-arguments
relocation target go.struct { main.T }.Mv not defined
undefined: "go.struct { main.T }.Mv"
```

The go tool

The go tool

- GOPATH optional since Go 1.8
- New: GOROOT based on executable path
- New: GOTMPDIR

Installing and using Go is simpler than ever: **download & extract** anywhere.

No need to set any environment variable! Most of the time.

Some rough edges, e.g., godoc does not infer GOROOT #23445.

go build

- Build caching ⚡

No need for `go build -i` (nor `go test -i`)

- Out-of-date packages

Now detected 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.

Removes the need for `go build -a`, most of the time.

go install

Now installs only the packages and commands listed directly on the command line.

Thanks to build caching, the very concept of installed packages may disappear in a future release.

go test

- Test result caching ⚡

```
$ go test crypto/rsa
ok      crypto/rsa      1.012s
$ go test crypto/rsa
ok      crypto/rsa      (cached)
```

Tests are cached only when **successful**, run with an **explicit list of packages** and using only a subset of the **-cpu**, **-list**, **-parallel**, **-run**, **-short**, and **-v** flags.

Files and **environment variables** used by the tests are checked.

The idiomatic way to bypass test caching is to use **-count=1**.

Under the hood of test caching

Go's src/internal/testlog/log.go uses an `atomic.Value` to store a "logger".

```
// Package testlog provides a back-channel communication path
// between tests and package os, so that cmd/go can see which
// environment variables and files a test consults.
package testlog

import "sync/atomic"

type Interface interface {
    Getenv(key string)
    Stat(file string)
    Open(file string)
    Chdir(dir string)
}

var logger atomic.Value
```

Under the hood of test caching

```
func SetLogger(impl Interface) {
    if logger.Load() != nil {
        panic("testlog: SetLogger must be called only once")
    }
    logger.Store(&impl)
}

func Logger() Interface {
    impl := logger.Load()
    if impl == nil {
        return nil
    }
    return *impl.(*Interface)
}

func Getenv(name string) {
    if log := Logger(); log != nil {
        log.Getenv(name)
    }
}
```

go test

- Includes go vet

```
$ GOPATH=$(go env GOROOT)/src/cmd/go/testdata
```

Before

```
$ go1.9.5 test vetpkg
ok      vetpkg  0.530s [no tests to run]
```

```
$ go1.9.5 vet vetpkg
/usr/local/Cellar/go/1.10.1/libexec/src/cmd/go/testdata/src/vetpkg/b.go:6: missing argument for
Printf("%d"): format reads arg 1, have only 0 args
exit status 1
```

Now

```
$ go test vetpkg
# vetpkg
/usr/local/Cellar/go/1.10.1/libexec/src/cmd/go/testdata/src/vetpkg/b.go:6: Printf format %d reads arg
#1, but call has only 0 args
FAIL    vetpkg [build failed]
```

go test

- Improved coverage reports

```
go test -coverpkg=all -coverprofile cover.out ./...
go test -coverpkg=crypto/... -coverprofile cover.out crypto/...
go tool cover -html cover.out
```

- Fail fast

```
go test -failfast
```

- JSON output

```
go test -json
```

go doc

Now functions returning []T or []*T group under type T, similar to functions returning T or *T:

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

type Address struct {
        Name    string // Proper name; may be empty.
        Address string // user@domain
}
    Address represents a single mail address. [...]


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

Standard Library

bytes

CL 74510

`Fields`, `FieldsFunc`, `Split`, and `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.

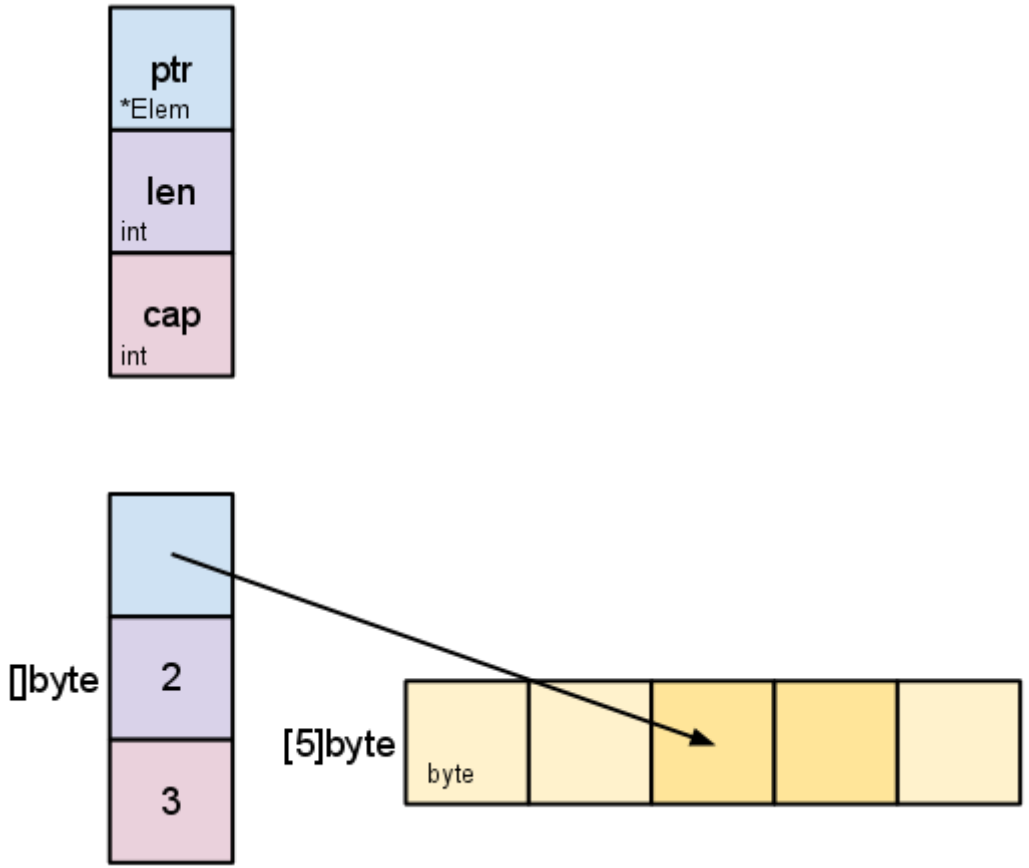
```
func Fields(s []byte) [][]byte
```

Example:

```
fmt.Printf("Fields are: %q", bytes.Fields([]byte("  foo bar  baz   ")))
// Output: Fields are: ["foo" "bar" "baz"]
```

Slices, length and capacity

blog.golang.org/go-slices-usage-and-internals



Slicing syntax

golang.org/ref/spec#Slice_expressions

Simple slice expressions

```
a[low : high]
```

Full slice expressions

```
a[low : high : max]
```

Slicing

```
s := []byte("abracadabra")
s1 := s[0:4]
s1 = append(s1, 'Z')
fmt.Printf("%s\n", s1)
fmt.Printf("%s\n", s) // abracadabra?

s = []byte("abracadabra")
s2 := s[0:4:4]
s2 = append(s2, 'Z')
fmt.Printf("%s\n", s2)
fmt.Printf("%s\n", s) // abracadabra?
```

Run

encoding/json

New Decoder method DisallowUnknownFields

```
type Rename struct {
    Old string
    New string
}

func main() {
    dec := json.NewDecoder(strings.NewReader(`
        {"old": "big", "new": "large"}
        {"old": "yesterday", "new": "tomorrow"}
        {"old": "sun", "new": "rain", "comment": "REVIEW"}
    `))
    dec.DisallowUnknownFields()
    for dec.More() {
        var r Rename
        err := dec.Decode(&r)
        fmt.Println(r, err)
    }
}
```

Run

math/rand

New Shuffle function and Rand.Shuffle method

```
func Shuffle(n int, swap func(i, j int))
func (r *Rand) Shuffle(n int, swap func(i, j int))
```

```
    rand.Seed(time.Now().UnixNano())

    s := []string{"Alice", "Bob", "Eve"}

    rand.Shuffle(len(s), func(i int, j int) {
        s[i], s[j] = s[j], s[i]
    })

    fmt.Println(s)
```

Run

flag

```
    s := flag.String("s", "", "this is a description\nthat spans multiple lines")
    x := flag.Int("x", 0, "short description")
```

Before 😕

```
$ go1.9.5 build -o program code/flags.go && ./program -h
Usage of ./program:
  -s string
        this is a description
that spans multiple lines
  -x int
        short description
```

Go 1.10 👍

```
$ go build -o program code/flags.go && ./program -h
Usage of ./program:
  -s string
        this is a description
        that spans multiple lines
  -x int
        short description
```

strings.Builder

Replacement for `bytes.Buffer` when accumulating text into a string.
Minimizes memory copying.

```
var b strings.Builder
for i := 3; i >= 1; i-- {
    fmt.Fprintf(&b, "%d...", i)
}
b.WriteString("Go Party!")
fmt.Println(b.String())
```

Run

net

The Conn and `Listener` implementations now guarantee that when `Close` returns, the underlying file descriptor has been closed.

```
l, err := net.Listen("tcp", ":2000")
if err != nil {
    log.Fatal(err)
}
l.Close()
l, err = net.Listen("tcp", ":2000") // the port should be free to bind again
...
```

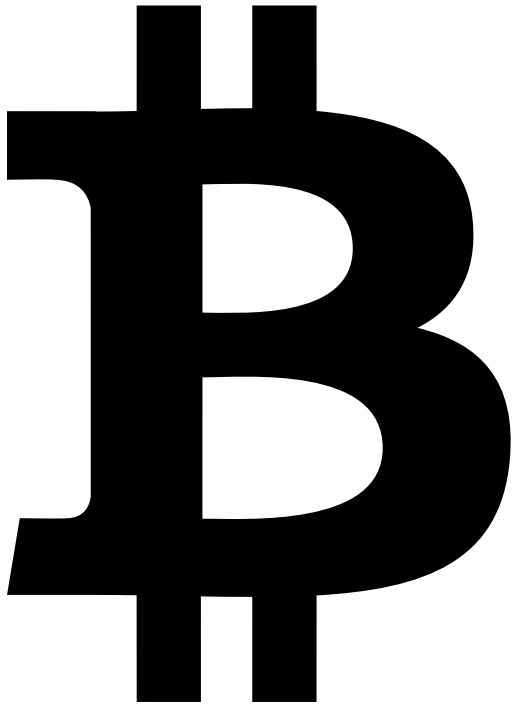
os

New `File` 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`.

Also matching `net.Conn`, `File`'s `Close` method now guarantee that when `Close` returns, the underlying file descriptor has been closed.

unicode



Go 1.10 supports Unicode 10.0

More

- See golang.org/doc/go1.10#library.
- Ports

No new supported operating systems nor processor architectures.
New assembler instructions and compiler improvements.
See golang.org/doc/go1.10#ports.

- Cgo

Fixed arbitrary code execution security hole. And more.
See golang.org/doc/go1.10#cgo.

- Performance

Most programs should run a bit faster, due to speedups in the garbage collector, better generated code, and optimizations in the core library.

Go 1.10.1

Go 1.10.1

go1.10.1 (released 2018/03/28) includes fixes to the compiler, runtime, and the archive/zip, crypto/tls, crypto/x509, encoding/json, net, net/http, and net/http/pprof packages.

See github.com/golang/go/issues?q=milestone%3AGo1.10.1

Recap

- Go1 Compatibility Promise
- Untyped Constants
- Method Expressions
- Build Caching
- Test Caching
- Slices
- strings.Builder
- ...

What's your favorite thing about Go & Go 1.10? Let's chat about it!

Misc

- godoc.org/golang.org/x/tools/present
- `present`-as-a-Service: talks.godoc.org
- Go Playground: play.golang.org
- godoc.org/golang.org/x/build/version

Thank you

Rodolfo Carvalho
AWS Elemental
rhcarvalho@gmail.com
http://rodolfocarvalho.net
https://dojo-brno.github.io