

[Discover Packages](#) > [Standard library](#) > [flag](#) 


flag

package

standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: [10](#) |Imported by: [258,113](#)



Details

 Valid [go.mod](#) file  Redistributable license  Tagged version  Stable version [Learn more](#)

Repository

[cs.opensource.google/go/go](#)

Links

 [Report a Vulnerability](#) Documentation 

<> Documentation

Overview

[Usage](#)[Command line flag syntax](#)

Package flag implements command-line flag parsing.

Usage

Define flags using `flag.String()`, `Bool()`, `Int()`, etc.

This declares an integer flag, `-n`, stored in the pointer `nFlag`, with type `*int`:

```
import "flag"
var nFlag = flag.Int("n", 1234, "help message for flag n")
```

If you like, you can bind the flag to a variable using the `Var()` functions.

```
var flagvar int
func init() {
    flag.IntVar(&flagvar, "flagname", 1234, "help message for flagname")
}
```

Or you can create custom flags that satisfy the `Value` interface (with pointer receivers) and couple them to flag parsing by

```
flag.Var(&flagVal, "name", "help message for flagname")
```

For such flags, the default value is just the initial value of the variable.

After all flags are defined, call

```
flag.Parse()
```

to parse the command line into the defined flags.

Flags may then be used directly. If you're using the flags themselves, they are all pointers; if you bind to variables, they're values.

```
fmt.Println("ip has value ", *ip)
fmt.Println("flagvar has value ", flagvar)
```

After parsing, the arguments following the flags are available as the slice `flag.Args()` or individually as `flag.Arg(i)`. The arguments are indexed from 0 through `flag.NArg()-1`.

Command line flag syntax

The following forms are permitted:

```
-flag
--flag    // double dashes are also permitted
-flag=x
-flag x    // non-boolean flags only
```

One or two dashes may be used; they are equivalent. The last form is not permitted for boolean flags because the meaning of the command

```
cmd -x *
```

where `*` is a Unix shell wildcard, will change if there is a file called `0`, `false`, etc. You must use the `-flag=false` form to turn off a boolean flag.

Flag parsing stops just before the first non-flag argument ("`-`" is a non-flag argument) or after the terminator "`--`".

Integer flags accept `1234`, `0664`, `0x1234` and may be negative. Boolean flags may be:

```
1, 0, t, f, T, F, true, false, TRUE, FALSE, True, False
```

Duration flags accept any input valid for `time.ParseDuration`.

The default set of command-line flags is controlled by top-level functions. The `FlagSet` type allows one to define independent sets of flags, such as to implement subcommands in a command-line interface. The methods of `FlagSet` are analogous to the top-level functions for the command-line flag set.

Index

Variables

```
func Arg(i int) string
func Args() []string
func Bool(name string, value bool, usage string) *bool
func BoolVar(p *bool, name string, value bool, usage string)
func Duration(name string, value time.Duration, usage string) *time.Duration
func DurationVar(p *time.Duration, name string, value time.Duration, usage string)
func Float64(name string, value float64, usage string) *float64
func Float64Var(p *float64, name string, value float64, usage string)
func Func(name, usage string, fn func(string) error)
func Int(name string, value int, usage string) *int
func Int64(name string, value int64, usage string) *int64
func Int64Var(p *int64, name string, value int64, usage string)
func IntVar(p *int, name string, value int, usage string)
func NArg() int
func NFlag() int
func Parse()
func Parsed() bool
func PrintDefaults()
func Set(name, value string) error
func String(name string, value string, usage string) *string
func StringVar(p *string, name string, value string, usage string)
func TextVar(p encoding.TextUnmarshaler, name string, value encoding.TextMarshaler, ...)
func Uint(name string, value uint, usage string) *uint
func Uint64(name string, value uint64, usage string) *uint64
func Uint64Var(p *uint64, name string, value uint64, usage string)
func UintVar(p *uint, name string, value uint, usage string)
func UnquoteUsage(flag *Flag) (name string, usage string)
func Var(value Value, name string, usage string)
func Visit(fn func(*Flag))
func VisitAll(fn func(*Flag))
type ErrorHandling
type Flag
    func Lookup(name string) *Flag
type FlagSet
    func NewFlagSet(name string, errorHandling ErrorHandling) *FlagSet
    func (f *FlagSet) Arg(i int) string
    func (f *FlagSet) Args() []string
    func (f *FlagSet) Bool(name string, value bool, usage string) *bool
    func (f *FlagSet) BoolVar(p *bool, name string, value bool, usage string)
    func (f *FlagSet) Duration(name string, value time.Duration, usage string) *time.Duration
    func (f *FlagSet) DurationVar(p *time.Duration, name string, value time.Duration, usage string)
```

```
func (f *FlagSet) ErrorHandler() ErrorHandler
func (f *FlagSet) Float64(name string, value float64, usage string) *float64
func (f *FlagSet) Float64Var(p *float64, name string, value float64, usage string)
func (f *FlagSet) Func(name, usage string, fn func(string) error)
func (f *FlagSet) Init(name string, errorHandler ErrorHandler)
func (f *FlagSet) Int(name string, value int, usage string) *int
func (f *FlagSet) Int64(name string, value int64, usage string) *int64
func (f *FlagSet) Int64Var(p *int64, name string, value int64, usage string)
func (f *FlagSet) IntVar(p *int, name string, value int, usage string)
func (f *FlagSet) Lookup(name string) *Flag
func (f *FlagSet) NArg() int
func (f *FlagSet) NFlag() int
func (f *FlagSet) Name() string
func (f *FlagSet) Output() io.Writer
func (f *FlagSet) Parse(arguments []string) error
func (f *FlagSet) Parsed() bool
func (f *FlagSet) PrintDefaults()
func (f *FlagSet) Set(name, value string) error
func (f *FlagSet) SetOutput(output io.Writer)
func (f *FlagSet) String(name string, value string, usage string) *string
func (f *FlagSet) StringVar(p *string, name string, value string, usage string)
func (f *FlagSet) TextVar(p encoding.TextUnmarshaler, name string, value encoding.TextMarshaler,
...)
func (f *FlagSet) Uint(name string, value uint, usage string) *uint
func (f *FlagSet) Uint64(name string, value uint64, usage string) *uint64
func (f *FlagSet) Uint64Var(p *uint64, name string, value uint64, usage string)
func (f *FlagSet) UintVar(p *uint, name string, value uint, usage string)
func (f *FlagSet) Var(value Value, name string, usage string)
func (f *FlagSet) Visit(fn func(*Flag))
func (f *FlagSet) VisitAll(fn func(*Flag))
```

type Getter

type Value

Examples

Package

Func

TextVar

Value

Constants

This section is empty.

Variables

```
var CommandLine = NewFlagSet(os.Args[0], ExitOnError)
```

CommandLine is the default set of command-line flags, parsed from os.Args. The top-level functions such as BoolVar, Arg, and so on are wrappers for the methods of CommandLine.

[View Source](#)

```
var ErrHelp = errors.New("flag: help requested")
```

ErrHelp is the error returned if the -help or -h flag is invoked but no such flag is defined.

[View Source](#)

```
var Usage = func() {  
    fmt.Fprintf(CommandLine.Output(), "Usage of %s:\n", os.Args[0])  
    PrintDefaults()  
}
```

Usage prints a usage message documenting all defined command-line flags to CommandLine's output, which by default is os.Stderr. It is called when an error occurs while parsing flags. The function is a variable that may be changed to point to a custom function. By default it prints a simple header and calls PrintDefaults; for details about the format of the output and how to control it, see the documentation for PrintDefaults. Custom usage functions may choose to exit the program; by default exiting happens anyway as the command line's error handling strategy is set to ExitOnError.

Functions

func Arg

```
func Arg(i int) string
```

Arg returns the i'th command-line argument. Arg(0) is the first remaining argument after flags have been processed. Arg returns an empty string if the requested element does not exist.

func Args

```
func Args() []string
```

Args returns the non-flag command-line arguments.

func Bool

```
func Bool(name string, value bool, usage string) *bool
```

Bool defines a bool flag with specified name, default value, and usage string. The return value is the address of a bool variable that stores the value of the flag.

func BoolVar

```
func BoolVar(p *bool, name string, value bool, usage string)
```

BoolVar defines a bool flag with specified name, default value, and usage string. The argument p points to a bool variable in which to store the value of the flag.

func Duration

```
func Duration(name string, value time.Duration, usage string) *time.Duration
```

Duration defines a time.Duration flag with specified name, default value, and usage string. The return value is the address of a time.Duration variable that stores the value of the flag. The flag accepts a value acceptable to time.ParseDuration.

func DurationVar

```
func DurationVar(p *time.Duration, name string, value time.Duration, usage string)
```

DurationVar defines a time.Duration flag with specified name, default value, and usage string. The argument p points to a time.Duration variable in which to store the value of the flag. The flag accepts a value acceptable to time.ParseDuration.

func Float64

```
func Float64(name string, value float64, usage string) *float64
```

Float64 defines a float64 flag with specified name, default value, and usage string. The return value is the address of a float64 variable that stores the value of the flag.

func Float64Var

```
func Float64Var(p *float64, name string, value float64, usage string)
```

Float64Var defines a float64 flag with specified name, default value, and usage string. The argument p points to a float64 variable in which to store the value of the flag.

func Func

added in go1.16

```
func Func(name, usage string, fn func(string) error)
```

Func defines a flag with the specified name and usage string. Each time the flag is seen, fn is called with the value of the flag. If fn returns a non-nil error, it will be treated as a flag value parsing error.

► [Example](#)

func Int

```
func Int(name string, value int, usage string) *int
```

Int defines an int flag with specified name, default value, and usage string. The return value is the address of an int variable that stores the value of the flag.

func Int64

```
func Int64(name string, value int64, usage string) *int64
```

Int64 defines an int64 flag with specified name, default value, and usage string. The return value is the address of an int64 variable that stores the value of the flag.

func Int64Var

```
func Int64Var(p *int64, name string, value int64, usage string)
```

Int64Var defines an int64 flag with specified name, default value, and usage string. The argument p points to an int64 variable in which to store the value of the flag.

func IntVar

```
func IntVar(p *int, name string, value int, usage string)
```

IntVar defines an int flag with specified name, default value, and usage string. The argument p points to an int variable in which to store the value of the flag.

func NArg

```
func NArg() int
```

NArg is the number of arguments remaining after flags have been processed.

func NFlag

```
func NFlag() int
```

NFlag returns the number of command-line flags that have been set.

func Parse

```
func Parse()
```

Parse parses the command-line flags from os.Args[1:]. Must be called after all flags are defined and before flags are accessed by the program.

func Parsed

```
func Parsed() bool
```

Parsed reports whether the command-line flags have been parsed.

func PrintDefaults

```
func PrintDefaults()
```

PrintDefaults prints, to standard error unless configured otherwise, a usage message showing the default settings of all defined command-line flags. For an integer valued flag x, the default output has the form

```
-x int
    usage-message-for-x (default 7)
```

The usage message will appear on a separate line for anything but a bool flag with a one-byte name. For bool flags, the type is omitted and if the flag name is one byte the usage message appears on the same line. The parenthetical default is omitted if the default is the zero value for the type. The listed type, here int, can be changed by placing a back-quoted name in the flag's usage string; the first such item in the message is taken to be a parameter name to show in the message and the back quotes are stripped from the message when displayed. For instance, given

```
flag.String("I", "", "search `directory` for include files")
```

the output will be

```
-I directory
    search directory for include files.
```

To change the destination for flag messages, call `CommandLine.SetOutput`.

func Set

```
func Set(name, value string) error
```

Set sets the value of the named command-line flag.

func String

```
func String(name string, value string, usage string) *string
```

String defines a string flag with specified name, default value, and usage string. The return value is the address of a string variable that stores the value of the flag.

func StringVar

```
func StringVar(p *string, name string, value string, usage string)
```


StringVar defines a string flag with specified name, default value, and usage string. The argument p points to a string variable in which to store the value of the flag.

func TextVar

added in go1.19

```
func TextVar(p encoding.TextUnmarshaler, name string, value encoding.TextMarshaler, usage string)
```

TextVar defines a flag with a specified name, default value, and usage string. The argument p must be a pointer to a variable that will hold the value of the flag, and p must implement encoding.TextUnmarshaler. If the flag is used, the flag value will be passed to p's UnmarshalText method. The type of the default value must be the same as the type of p.

► Example

func Uint

```
func Uint(name string, value uint, usage string) *uint
```

Uint defines a uint flag with specified name, default value, and usage string. The return value is the address of a uint variable that stores the value of the flag.

func Uint64

```
func Uint64(name string, value uint64, usage string) *uint64
```

Uint64 defines a uint64 flag with specified name, default value, and usage string. The return value is the address of a uint64 variable that stores the value of the flag.

func Uint64Var

```
func Uint64Var(p *uint64, name string, value uint64, usage string)
```

Uint64Var defines a uint64 flag with specified name, default value, and usage string. The argument p points to a uint64 variable in which to store the value of the flag.

func UintVar

```
func UintVar(p *uint, name string, value uint, usage string)
```

UintVar defines a uint flag with specified name, default value, and usage string. The argument p points to a uint variable in which to store the value of the flag.

func UnquoteUsage

added in go1.5

```
func UnquoteUsage(flag *Flag) (name string, usage string)
```

UnquoteUsage extracts a back-quoted name from the usage string for a flag and returns it and the unquoted usage. Given "a `name` to show" it returns ("name", "a name to show"). If there are no back quotes, the name is an educated guess of the type of the flag's value, or the empty string if the flag is boolean.

func Var

```
func Var(value Value, name string, usage string)
```

Var defines a flag with the specified name and usage string. The type and value of the flag are represented by the first argument, of type Value, which typically holds a user-defined implementation of Value. For instance, the caller could create a flag that turns a comma-separated string into a slice of strings by giving the slice the methods of Value; in particular, Set would decompose the comma-separated string into the slice.

func Visit

```
func Visit(fn func(*Flag))
```

Visit visits the command-line flags in lexicographical order, calling fn for each. It visits only those flags that have been set.

func VisitAll

```
func VisitAll(fn func(*Flag))
```

VisitAll visits the command-line flags in lexicographical order, calling fn for each. It visits all flags, even those not set.

Types

type ErrorHandling

```
type ErrorHandling int
```

ErrorHandling defines how FlagSet.Parse behaves if the parse fails.

```
const (
    ContinueOnError ErrorHandling = iota // Return a descriptive error.
    ExitOnError                          // Call os.Exit(2) or for -h/-help Exit(0).
    PanicOnError                        // Call panic with a descriptive error.
)
```

These constants cause FlagSet.Parse to behave as described if the parse fails.

type Flag

```

type Flag struct {
    Name      string // name as it appears on command line
    Usage     string // help message
    Value     Value  // value as set
    DefValue  string // default value (as text); for usage message
}

```

A Flag represents the state of a flag.

func Lookup

```

func Lookup(name string) *Flag

```

Lookup returns the Flag structure of the named command-line flag, returning nil if none exists.

type FlagSet

```

type FlagSet struct {
    // Usage is the function called when an error occurs while parsing flags.
    // The field is a function (not a method) that may be changed to point to
    // a custom error handler. What happens after Usage is called depends
    // on the ErrorHandler setting; for the command line, this defaults
    // to ExitOnError, which exits the program after calling Usage.
    Usage func()
    // contains filtered or unexported fields
}

```

A FlagSet represents a set of defined flags. The zero value of a FlagSet has no name and has ContinueOnError error handling.

Flag names must be unique within a FlagSet. An attempt to define a flag whose name is already in use will cause a panic.

func NewFlagSet

```

func NewFlagSet(name string, errorHandler ErrorHandler) *FlagSet

```

NewFlagSet returns a new, empty flag set with the specified name and error handling property. If the name is not empty, it will be printed in the default usage message and in error messages.

func (*FlagSet) Arg

```

func (f *FlagSet) Arg(i int) string

```

Arg returns the i'th argument. Arg(0) is the first remaining argument after flags have been processed. Arg returns an empty string if the requested element does not exist.

func (*FlagSet) Args

```
func (f *FlagSet) Args() []string
```

Args returns the non-flag arguments.

func (*FlagSet) Bool

```
func (f *FlagSet) Bool(name string, value bool, usage string) *bool
```

Bool defines a bool flag with specified name, default value, and usage string. The return value is the address of a bool variable that stores the value of the flag.

func (*FlagSet) BoolVar

```
func (f *FlagSet) BoolVar(p *bool, name string, value bool, usage string)
```

BoolVar defines a bool flag with specified name, default value, and usage string. The argument p points to a bool variable in which to store the value of the flag.

func (*FlagSet) Duration

```
func (f *FlagSet) Duration(name string, value time.Duration, usage string) *time.Duration
```

Duration defines a time.Duration flag with specified name, default value, and usage string. The return value is the address of a time.Duration variable that stores the value of the flag. The flag accepts a value acceptable to time.ParseDuration.

func (*FlagSet) DurationVar

```
func (f *FlagSet) DurationVar(p *time.Duration, name string, value time.Duration, usage string)
```

DurationVar defines a time.Duration flag with specified name, default value, and usage string. The argument p points to a time.Duration variable in which to store the value of the flag. The flag accepts a value acceptable to time.ParseDuration.

func (*FlagSet) ErrorHandler

added in go1.10

```
func (f *FlagSet) ErrorHandler() ErrorHandler
```

ErrorHandler returns the error handling behavior of the flag set.

func (*FlagSet) Float64

```
func (f *FlagSet) Float64(name string, value float64, usage string) *float64
```

Float64 defines a float64 flag with specified name, default value, and usage string. The return value is the address of a float64 variable that stores the value of the flag.

func (*FlagSet) Float64Var

```
func (f *FlagSet) Float64Var(p *float64, name string, value float64, usage string)
```

Float64Var defines a float64 flag with specified name, default value, and usage string. The argument p points to a float64 variable in which to store the value of the flag.

func (*FlagSet) Func

added in go1.16

```
func (f *FlagSet) Func(name, usage string, fn func(string) error)
```

Func defines a flag with the specified name and usage string. Each time the flag is seen, fn is called with the value of the flag. If fn returns a non-nil error, it will be treated as a flag value parsing error.

func (*FlagSet) Init

```
func (f *FlagSet) Init(name string, errorHandler ErrorHandling)
```

Init sets the name and error handling property for a flag set. By default, the zero FlagSet uses an empty name and the ContinueOnError error handling policy.

func (*FlagSet) Int

```
func (f *FlagSet) Int(name string, value int, usage string) *int
```

Int defines an int flag with specified name, default value, and usage string. The return value is the address of an int variable that stores the value of the flag.

func (*FlagSet) Int64

```
func (f *FlagSet) Int64(name string, value int64, usage string) *int64
```

Int64 defines an int64 flag with specified name, default value, and usage string. The return value is the address of an int64 variable that stores the value of the flag.

func (*FlagSet) Int64Var

```
func (f *FlagSet) Int64Var(p *int64, name string, value int64, usage string)
```

Int64Var defines an int64 flag with specified name, default value, and usage string. The argument p points to an int64 variable in which to store the value of the flag.

func (*FlagSet) IntVar

```
func (f *FlagSet) IntVar(p *int, name string, value int, usage string)
```

IntVar defines an int flag with specified name, default value, and usage string. The argument p points to an int variable in which to store the value of the flag.

func (*FlagSet) Lookup

```
func (f *FlagSet) Lookup(name string) *Flag
```

Lookup returns the Flag structure of the named flag, returning nil if none exists.

func (*FlagSet) NArg

```
func (f *FlagSet) NArg() int
```

NArg is the number of arguments remaining after flags have been processed.

func (*FlagSet) NFlag

```
func (f *FlagSet) NFlag() int
```

NFlag returns the number of flags that have been set.

func (*FlagSet) Name

added in go1.10

```
func (f *FlagSet) Name() string
```

Name returns the name of the flag set.

func (*FlagSet) Output

added in go1.10

```
func (f *FlagSet) Output() io.Writer
```

Output returns the destination for usage and error messages. os.Stderr is returned if output was not set or was set to nil.

func (*FlagSet) Parse

```
func (f *FlagSet) Parse(arguments []string) error
```

Parse parses flag definitions from the argument list, which should not include the command name. Must be called after all flags in the FlagSet are defined and before flags are accessed by the program. The return value will be ErrHelp if -help or -h were set but not defined.

func (*FlagSet) Parsed

```
func (f *FlagSet) Parsed() bool
```

Parsed reports whether f.Parse has been called.

func (*FlagSet) PrintDefaults

```
func (f *FlagSet) PrintDefaults()
```

PrintDefaults prints, to standard error unless configured otherwise, the default values of all defined command-line flags in the set. See the documentation for the global function PrintDefaults for more information.

func (*FlagSet) Set

```
func (f *FlagSet) Set(name, value string) error
```

Set sets the value of the named flag.

func (*FlagSet) SetOutput

```
func (f *FlagSet) SetOutput(output io.Writer)
```

SetOutput sets the destination for usage and error messages. If output is nil, os.Stderr is used.

func (*FlagSet) String

```
func (f *FlagSet) String(name string, value string, usage string) *string
```

String defines a string flag with specified name, default value, and usage string. The return value is the address of a string variable that stores the value of the flag.

func (*FlagSet) StringVar

```
func (f *FlagSet) StringVar(p *string, name string, value string, usage string)
```

StringVar defines a string flag with specified name, default value, and usage string. The argument p points to a string variable in which to store the value of the flag.

func (*FlagSet) TextVar

added in go1.19

```
func (f *FlagSet) TextVar(p encoding.TextUnmarshaler, name string, value encoding.TextMarshaler, usage string)
```

TextVar defines a flag with a specified name, default value, and usage string. The argument p must be a pointer to a variable that will hold the value of the flag, and p must implement encoding.TextUnmarshaler. If the flag is used, the flag value will be passed to p's UnmarshalText method. The type of the default value must be the same as the type of p.

func (*FlagSet) Uint

```
func (f *FlagSet) Uint(name string, value uint, usage string) *uint
```

Uint defines a uint flag with specified name, default value, and usage string. The return value is the address of a uint variable that stores the value of the flag.

func (*FlagSet) Uint64

```
func (f *FlagSet) Uint64(name string, value uint64, usage string) *uint64
```

Uint64 defines a uint64 flag with specified name, default value, and usage string. The return value is the address of a uint64 variable that stores the value of the flag.

func (*FlagSet) Uint64Var

```
func (f *FlagSet) Uint64Var(p *uint64, name string, value uint64, usage string)
```

Uint64Var defines a uint64 flag with specified name, default value, and usage string. The argument p points to a uint64 variable in which to store the value of the flag.

func (*FlagSet) UintVar

```
func (f *FlagSet) UintVar(p *uint, name string, value uint, usage string)
```

UintVar defines a uint flag with specified name, default value, and usage string. The argument p points to a uint variable in which to store the value of the flag.

func (*FlagSet) Var

```
func (f *FlagSet) Var(value Value, name string, usage string)
```

Var defines a flag with the specified name and usage string. The type and value of the flag are represented by the first argument, of type Value, which typically holds a user-defined implementation of Value. For instance, the caller could create a flag that turns a comma-separated string into a slice of strings by giving the slice the methods of Value; in particular, Set would decompose the comma-separated string into the slice.

func (*FlagSet) Visit

```
func (f *FlagSet) Visit(fn func(*Flag))
```

Visit visits the flags in lexicographical order, calling fn for each. It visits only those flags that have been set.

func (*FlagSet) VisitAll

```
func (f *FlagSet) VisitAll(fn func(*Flag))
```

VisitAll visits the flags in lexicographical order, calling fn for each. It visits all flags, even those not set.

type Getter

added in go1.2


```
type Getter interface {  
    Value  
    Get() any  
}
```

Getter is an interface that allows the contents of a Value to be retrieved. It wraps the Value interface, rather than being part of it, because it appeared after Go 1 and its compatibility rules. All Value types provided by this package satisfy the Getter interface, except the type used by Func.

type Value

```
type Value interface {  
    String() string  
    Set(string) error  
}
```

Value is the interface to the dynamic value stored in a flag. (The default value is represented as a string.)

If a Value has an IsBoolFlag() bool method returning true, the command-line parser makes -name equivalent to -name=true rather than using the next command-line argument.

Set is called once, in command line order, for each flag present. The flag package may call the String method with a zero-valued receiver, such as a nil pointer.

► [Example](#)



Source Files

[View all](#) 

[flag.go](#)

Why Go

[Use Cases](#)

[Case Studies](#)

Get Started

[Playground](#)

[Tour](#)

[Stack Overflow](#)

[Help](#)

Packages

[Standard Library](#)

[About Go Packages](#)

About

[Download](#)

[Blog](#)

[Issue Tracker](#)

[Release Notes](#)

[Brand Guidelines](#)

[Code of Conduct](#)

Connect

[Twitter](#)

[GitHub](#)

[Slack](#)

[r/golang](#)

[Meetup](#)

[Golang Weekly](#)

[Copyright](#)

[Terms of Service](#)

[Privacy Policy](#)

[Report an Issue](#)



[Google](#)