Go Cheat Sheet

Last modified: 2023-09-30

Tooling

`go env`	Show all Go settings
`go help environment`	Show purpose of all Go settings

Development

`go run <dir>`	Runs executable in directory `<dir>`
`go get <path>@<ver>`	Fetch dependency `<path>` optionally at version `<ver>`
`go list -m all`	Show all dependencies
`go mod why -m <path>`	Why does this module depend on `<path>`?
`go clean -modcache`	Clean the module cache
`gofmt -d -w -r 'abc -> xyz'`	Replace `abc` with `xyz`
`go doc <name>`	Display summary documentation for `<name>`
`go doc -all <name>`	Display full documentation for `<name>`
`go doc -src <name>`	Display source code for `<name>`

Testing

`go test <dir>`	Runs all tests in directory `<dir>`
`go test <dir>/...`	Runs all tests in `<dir>` and subdirectories
`go test -race …`	Run tests with data race detector
`go test -count=<n> …`	Bypass test cache and run tests `<n>` times
`go clean -testcache`	Clean up all cached test results
`go test -v -run=<regex> …`	Run all tests matching regular expression `<regex>`
`go test -short …`	Skips tests flagged as long-running
`go test -failfast …`	Stop running tests on first failure
`go test -fuzz …`	Run any defined fuzz tests until a failure is found or aborted
`go test -fuzz -fuzztime=<dur> …`	Run any defined fuzz tests until a failure is found or `<dur>` expires
`go test -cover …`	Show code coverage summary
`go test -coverprofile=<path>`	Generate coverage metrics in output file `<path>`
`go test -covermode=count …`	Additionally capture the number of times each line is hit
`go tool cover -html=<path>`	Show report from recorded coverage metrics in browser
`go tool cover -func=<path>`	Show report from recorded coverage metrics in terminal

Tidying Up

`go fmt ./...`	Reformat this and all subdirectories
`go vet <path>`	Run static code analysis on file or directory `<path>`
`go mod tidy`	Remove unused dependencies
`go mod verify`	Verify that dependencies are still clean

Building

`go build <dir> -o=<path>`	Build package in `<dir>` optionally placing output in `<path>`
`go build -a …`	Bypass cache and rebuild all packages
`go clean -cache`	Clean entire build cache
`GOOS=linux GOARCH=amd64 go build …`	Specify operating system and architecture for cross-compilation
`go tool dist list`	List all valid OS/arch targets for this compiler
`go build -gcflags="<flags>" …`	Pass `<flags>` to the compiler for packages listed on command-line
`… -gcflags="<patt>=<flags>" …`	Pass `<flags>` to the compiler for all packages matching <patt>
`go build -ldflags="<flags>" …`	Pass `<flags>` to the linker

Ongoing Maintenance

`go test -bench=<dir>`	Runs tests and benchmarks from `<dir>`
`… -benchmem`	Shows memory allocations for each iteration as well as execution time
`… -benchtime=<arg>`	Sets the threshold time (e.g. `5s`) or number of iterations (e.g. `500x`)
`… -count=<n>`	Repeats each benchmark <n> times in sequence
`… -cpu=<ns>`	Runs each benchmark with `GOMAXPROCS` at each value in list `<ns>`
`go test -cpuprofile=<file>`	Run tests outputting CPU time profiling data to `<file>`
`go test -memprofile=<file>`	Run tests outputting memory allocation profiling data to `<file>`
`go test -blockprofile=<file>`	Run tests outputting blocking call profiling data to `<file>`
`go test -mutexprofile=<file>`	Run tests outputting mutex contention profiling data to `<file>`
`go tool pprof <binary> <file>`	Run interactive CLI on executable `<binary>` with profiling data `<file>`
`… -top -functions …`	Show top users at granularity of functions
`… -top -lines …`	Show top users at granularity of source code lines
`… -http localhost:<port> …`	Start interactive web interface on `<port>` and open a browser on it
`go list -u -m all`	For each dependency, show current version and upstream latest
`go get <path>@<ver>`	Also used to update existing dependency version
`go mod tidy && go test all`	Good idea to do this after updating dependencies

Last modified: 2023-07-08

Scalar Types

```
bool (true / false)
```

```
string
```

```
float32          float64
complex64        complex128
```

```
uint   uint8   uint16
uint32   uint64
```

```
int   int8   int16
int32   int64
```

```
uintptr
```

```
rune (int32)    byte (uint8)
```

Pointer & Channel Types

```
var ptr *int
```

```
var value int = 99      ptr = &value      value = *ptr
```

```
unbuffered := make(chan int)
```

```
buffered := make(chan int, 100)
```

```
chan<- int    (sendable)
```

```
<-chan int    (receivable)
```

```
chan int      (both)
```

Arrays, Slices & Maps

```
var a [100]uint32
var b = [...]float32{1.2, 3.4}
c := [“one”, “two”, “three”]
```

```
var d []uint32          f := append(d, item, …)
d = a[5:22]             numCopied := copy(dst, src)
e := make([]int, 5)
```

```
x := map[string]int{“one”: 1}          x[“two”] = 2
y := make(map[string]int, 100)         delete(x, “one”)
```

Structs

```
type Person struct {
      name string
      age uint
}
```

```
jedi := Person {
      name: “Yoda”,
      age: 800,
}
```

```
n := jedi.name
jedi.age = 900
```

```
type Student struct {
      Person
      entryYear uint
}
```

```
andy := Student {
      Person: Person {
            name: “Andy”,
            age: 20,
      },
      entryYear: 2020,
}
```

```
n := andy.age
andy.age = 18
andy.entryYear = 2023
```

Constants

```
const name string = “Andy”    (typed)
const name = “Andy”           (untyped)
```

```
const (
      KB SizeUnit = 1 << (10 * (iota + 1))
      MB
… )
```

Branching

```
if x > y { …
} else if x == y { …
} else { …
}
```

```
switch n {
case 1: …
      fallthrough
case 2: …
}
```

```
switch n {
case n < 100: …
case n > 500: …
default: …
}
```

```
switch n(.type) {
case uint32: …
case int32: …
case nil: …
}
```

Loops

```
for i > 0 { …
      i -= 1
}
```

```
for { …
      break
}
```

```
for i := 0; i < 10; i++ {
      …
}
```

```
for i, val := range items {
      …
}
```

Functions & Closures

```
func name(arg1 int, arg2 string) int {
      … return 123
```

```
func name(arg string) (string, error) {
      … return “”, errors.New(“Foo”)
```

```
func name(arg int, more ...string) {
      for i, value := range more { … }
      anotherFunc(arg, more...)
}
```

```
func name() (someVar int) {
      someVar = 123
      return
}
```

```
func ticketMachine(start int) func() int {
      return func() int {
            start++
            return start
      }
}
```

Panic, Defer & Recover

```
panic(value)
```

```
func someFunc(arg int) { …
}
Func otherFunc() {
      defer someFunc(123) …
}
```

```
func name() {
      defer func() {
            if r := recover(); r != nil {
                  …
            }
      }()
      …
}
```

Predeclared Identifiers

```
any bool byte comparable error
complex64 complex128 float32 float64
int int8 int16 int32 int64 rune string
uint uint8 uint16 uint32 uint64 uintptr
```

```
true false iota nil
```

```
append cap close complex copy delete imag len
make new panic print println real recover
```

Methods

```
type Point struct {
    x, y float64
}
func (p *Point) move(dx, dy float64) {
    p.x += dx
    p.y += dy
}
```

Value type	Receiver type	Callable?
T	T	✓
T	*T	
*T	T	✓
*T	*T	✓

Interfaces

```
type Shape interface {
    area() float64
    translate(x, y float64)
}
```

```
func f(s Shape) float64 {
      s.translate(1.0, 5.0)
      return s.area()
}
```

```
type Cornered interface {
    Shape
    numCorners() int
}
```

All types implement the empty interface

```
interface{}
```

The ~ means "same underlying type", without it's an exact match

```
type SignedInt interface {
    ~int | ~int8 | ~int16 | ~int32 | ~int64
}
```

Generics

```
type List[T any] struct {
      …
}
```

```
func (node *List[T]) insertAfter(item T) *List[T] {
      …
}
```

```
func find[T comparable](n T, h []T) int {
      …
}
```

```
func newList[T any](item T) *List[T] {
    return &List[T]{item, nil, nil}
}
```

Go Routines & Channels

```
func consumer(c <-chan int) {
      for i := range c { … }
}
```

```
func producer(c chan<- int) {
      for i := 0; i < 10; i++ {
            c <- i
      }
      close(c)
}
```

```
func main() {
      c := make(chan int)
      go consumer(c)
      go producer(c)
… }
```

```
func consumer(c <-chan int) {
      for {
            i, more := <- c
            if !more {
                  break
            }
      … }
}
```

```
select {
      case m := <-c1: …
      case m := <-c2: …
      case m := <-c3: …
      case <-time.After(…): …
}
```

Blocking with timeout

```
select {
      case m := <-c1: …
      case m := <-c2: …
      case m := <-c3: …
      default:
            // No msgs
            …
}
```

Non-blocking

Go Cheat Sheet

Last modified: 2023-07-08

Operators

u	`+` ≡ `0+x`	`-` ≡ `0-x`	`!` NOT	`^` Bitwise	`*` Dereference	`&` Address of	`<-` Channel receive
5	`*` Product	`/` Quotient	`%` Remainder	`<<` `>>` Left/right shift	`&` Bitwise AND	`&^` Bit clear (AND NOT)	
4	`+` Add	`-` Subtract	`|` Bitwise OR	`^` Bitwise XOR			
3	`==` Equal	`!=` Not Equal	`<` Less	`<=` Less or equal	`>` Greater	`>=` Greater or equal	
2	`&&` Logical AND						
1	`||` Logical OR						

Unary operators are always highest precedence. Within a precedence, binary operators are left associative.

Channel send `<-` *as well as unary* `++` *and* `--` *are statements not operators.*
e.g. `*p++` *is equivalent to* `(*p)++`

fmt Codes

`%` flags width `.` precision code

Everything is optional except `%` *and* code.

General

%v	Value in default format
%+v	Add field names in structs
%#v	Value in Go syntax
%T	Go syntax for value *type*
%%	Literal % symbol

Boolean

%t	Either `true` or `false`

String & Byte Slice

%s	Uninterpreted bytes of string
%q	Double-quoted string literal
%x	Bytes in hex, lowercase
%X	Uppercase version of %x

Flags

+	Print sign even on positive values
-	Left-justify instead of right-justify
#	Alternate format (varies by code)
SP	Leave a space for an elided sign
0	Pad with zeroes rather than spaces

Integers

%b	Value in binary
%c	Unicode code point character
%d	Value in decimal
%o	Value in octal
%0	Value in octal with `0o` prefix
%q	Single-quoted character literal
%x	Base 16 using lowercase letters
%X	Base 16 using uppercase letters
%U	Unicode format (i.e. "U+%04X")

Floating Point & Complex

%b	Decimalless scientific notation
%e	Scientific notation (`1.2e+12`)
%E	Uppercase version of %e
%f	Decimal form with no exponent
%F	Alias for %f
%g	Choose %e or %f based on size
%G	Choose %E or %F based on size
%x	Hex notation, power 2 exponent
%X	Uppercase version of %x

Slice

%p	Address of `x[0]` as for pointer

Pointer

%p	Address in hex with leading `0x`

Format for %v

`bool`	%t
`int, …`	%d
`uint, …`	%d
`float32, …`	%g
`complex64, …`	%g
`string`	%s
`chan`	%p
Pointer	%p

Format Functions

```
Appendf([]byte, string, …) []byte
Errorf(string, …) error
Fprintf(io.Writer, string, …)(i…, e…)
Fscanf(io.Reader, string, …) (i…, e…)
Printf(string, …) (int, error)
Scanf(string, …) (int, error)
Sprintf(string, …) string
Sscanf(string, string, …) (int, e…)
```