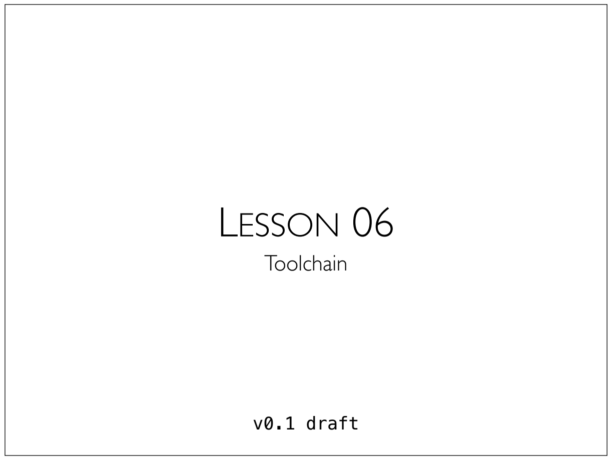
Learn the Go Programming Language

For experienced developers or
those of an adventurous nature

gotutorial.net
@GoTutorialNet

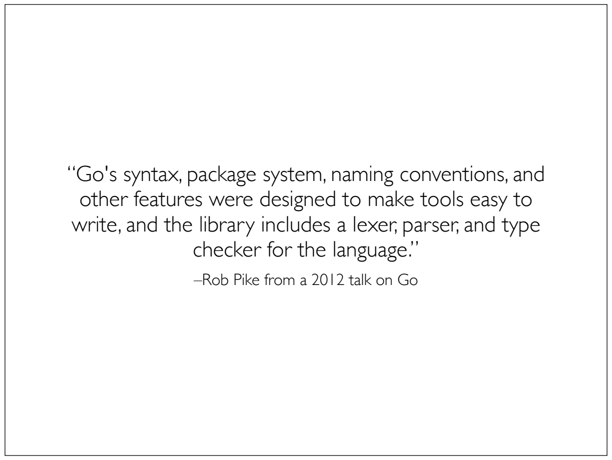
Matt Nunogawa
@amattn



Lesson 06

Toolchain

v0.1 draft

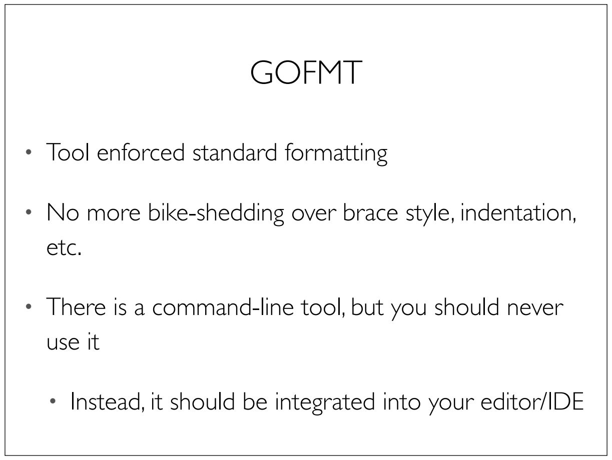


"Go's syntax, package system, naming conventions, and other features were designed to make tools easy to write, and the library includes a lexer, parser, and type checker for the language."

–Rob Pike from a 2012 talk on Go

http://talks.golang.org/2012/splash.article#TOC_17

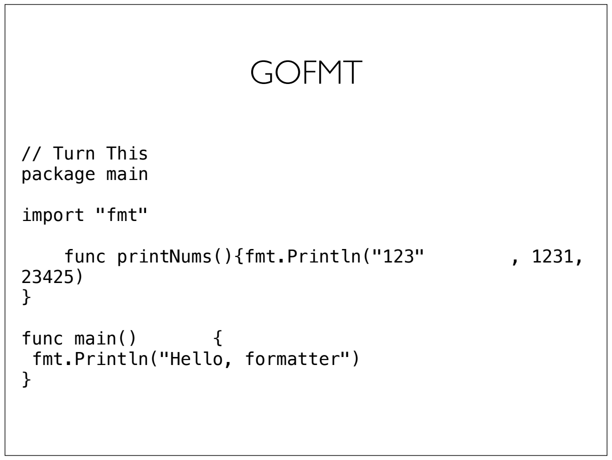
Formatting



GOFMT

- Tool enforced standard formatting
- No more bike-shedding over brace style, indentation, etc.
- There is a command-line tool, but you should never use it
  - Instead, it should be integrated into your editor/IDE

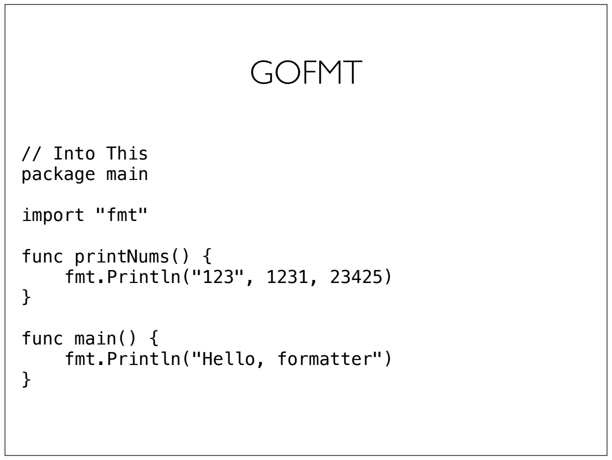
http://bikeshed.com



GOFMT

```
// Turn This
package main

import "fmt"

    func printNums(){fmt.Println("123"        , 1231,
23425)
}

func main()       {
 fmt.Println("Hello, formatter")
}
```

http://play.golang.org/p/JUHbccsqFx

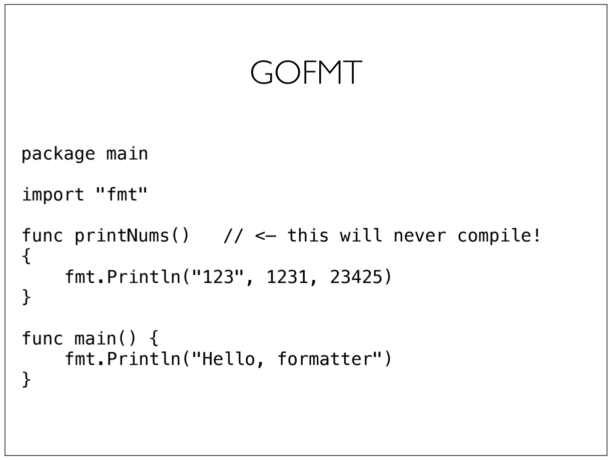


GOFMT

```
// Into This
package main

import "fmt"

func printNums() {
    fmt.Println("123", 1231, 23425)
}

func main() {
    fmt.Println("Hello, formatter")
}
```

http://play.golang.org/p/JUHbccsqFx

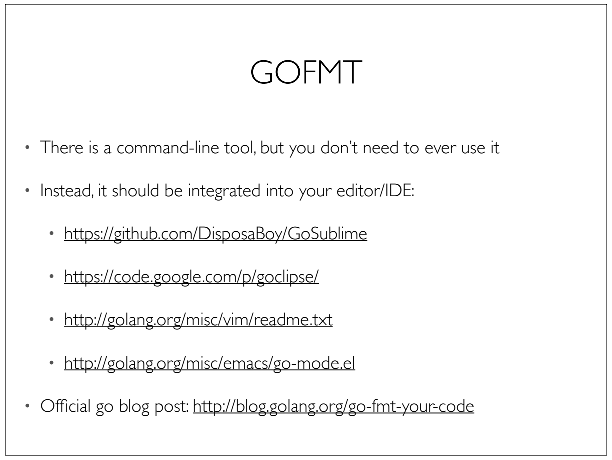
press the format button



GOFMT

```go
package main

import "fmt"

func printNums()   // <– this will never compile!
{
    fmt.Println("123", 1231, 23425)
}

func main() {
    fmt.Println("Hello, formatter")
}
```

http://bikeshed.com



GOFMT

- There is a command-line tool, but you don't need to ever use it
- Instead, it should be integrated into your editor/IDE:
  - https://github.com/DisposaBoy/GoSublime
  - https://code.google.com/p/goclipse/
  - http://golang.org/misc/vim/readme.txt
  - http://golang.org/misc/emacs/go-mode.el
- Official go blog post: http://blog.golang.org/go-fmt-your-code

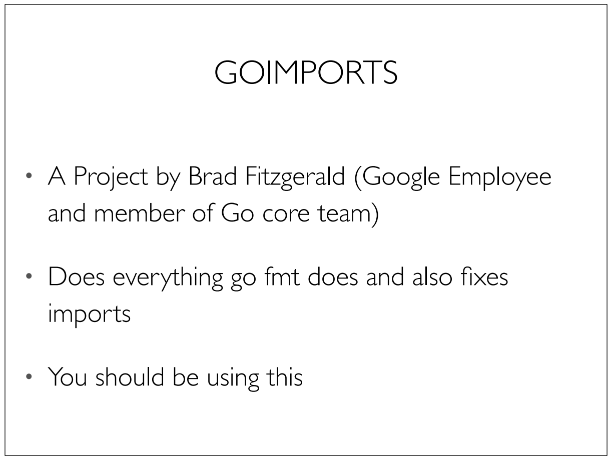
https://github.com/DisposaBoy/GoSublime

https://code.google.com/p/goclipse/

http://golang.org/misc/vim/readme.txt

http://golang.org/misc/emacs/go-mode.el

http://blog.golang.org/go-fmt-your-code



GOIMPORTS

- A Project by Brad Fitzgerald (Google Employee and member of Go core team)
- Does everything go fmt does and also fixes imports
- You should be using this

https://github.com/bradfitz/goimports

GOCODE

- Codesense tool
- https://github.com/nsf/gocode
- Gives generic code completion and metadata around names
- Similar to gofmt, there exist integrations for most major editors/IDEs

Compiling, Linking & Building

But first a digression
$GOPATH

$GOPATH

- "An environment variable that lists places to look for go code"

```
go help gopath
```

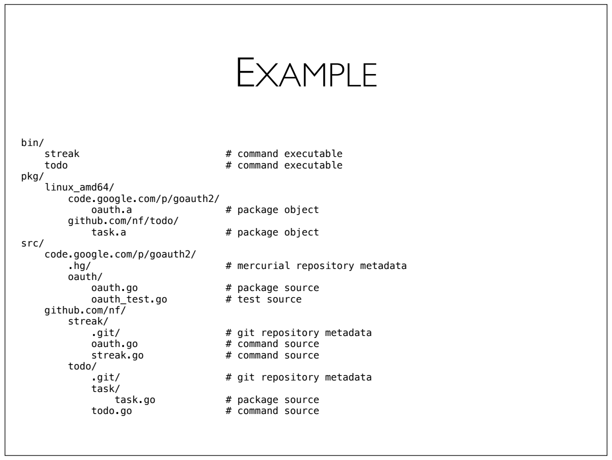
$GOPATH HAS A DEFINED LAYOUT

- “An environment variable that lists places to look for go code”

```
GOPATH=“/home/amattn/gopath”

/home/amattn/gopath/
/home/amattn/gopath/src
/home/amattn/gopath/bin
/home/amattn/gopath/pkg
```

$GOPATH HAS A DEFINED LAYOUT

- bin/ contains the compiled executables
- pkg/ contains the compiled object files (.a)
- src/ is organized by package namespace, including repo (github, code.google.com, etc)



Example

```
bin/
    streak                          # command executable
    todo                            # command executable
pkg/
    linux_amd64/
        code.google.com/p/goauth2/
            oauth.a                 # package object
        github.com/nf/todo/
            task.a                  # package object
src/
    code.google.com/p/goauth2/
        .hg/                        # mercurial repository metadata
        oauth/
            oauth.go                # package source
            oauth_test.go           # test source
    github.com/nf/
        streak/
            .git/                   # git repository metadata
            oauth.go                # command source
            streak.go               # command source
        todo/
            .git/                   # git repository metadata
            task/
                task.go             # package source
            todo.go                 # command source
```

http://golang.org/doc/code.html

Usage Models

- One GOPATH for everything
  - Simpler conceptually
  - You need to worry about dependencies, vendoring and versioning
- One GOPATH per project
  - more like virtualenv-style management
  - putting your GOPATH under version control works pretty well as long as you like git submodules

Compiling, Linking & Building

THE OLD WAY

```
# compile the go code
6g somecode.go
```

produces somecode.6

```
# link the go code
6l somecode.6
```

produces 6.out

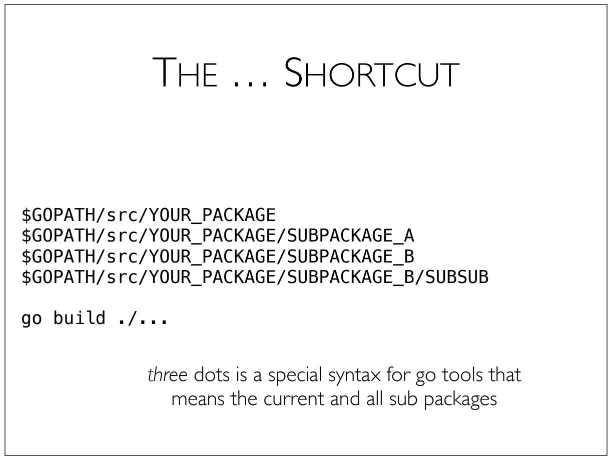
```
# link the go code
6l –o someexec somecode.6
```

produces someexec

THE NEW WAY

```
cd $GOPATH/src/YOUR_PACKAGE

go build
```

produces YOUR_PACKAGE



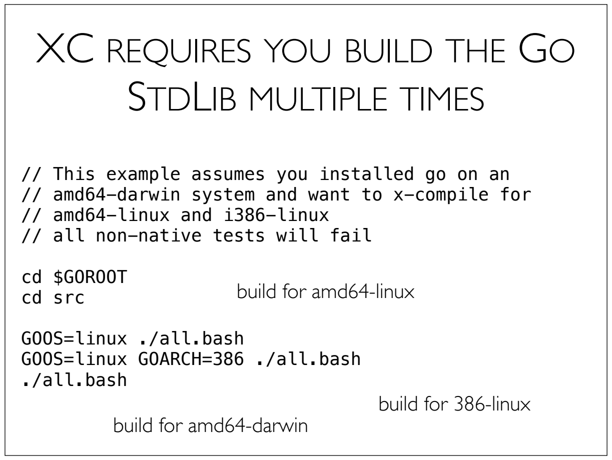
The ... Shortcut

```
$GOPATH/src/YOUR_PACKAGE
$GOPATH/src/YOUR_PACKAGE/SUBPACKAGE_A
$GOPATH/src/YOUR_PACKAGE/SUBPACKAGE_B
$GOPATH/src/YOUR_PACKAGE/SUBPACKAGE_B/SUBSUB

go build ./...
```

three dots is a special syntax for go tools that means the current and all sub packages

watch out for text autocorrect tools that replace three periods with the unicode ellipses character

Cross-Compiling



XC REQUIRES YOU BUILD THE GO STDLIB MULTIPLE TIMES

```
// This example assumes you installed go on an
// amd64-darwin system and want to x-compile for
// amd64-linux and i386-linux
// all non-native tests will fail

cd $GOROOT
cd src

GOOS=linux ./all.bash
GOOS=linux GOARCH=386 ./all.bash
./all.bash
```

build for amd64-linux

build for 386-linux

build for amd64-darwin

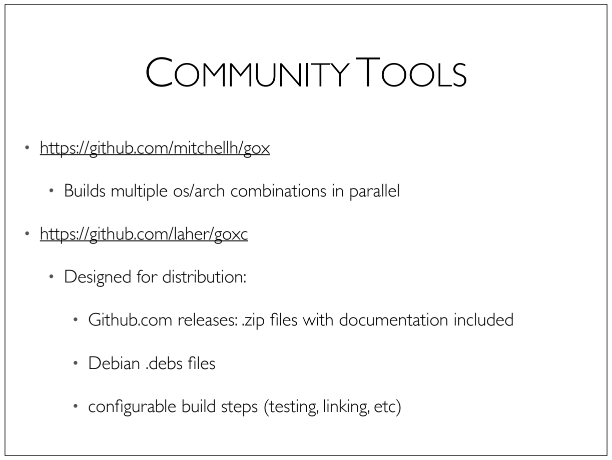
non-native tests will fail. no biggie

Updating Go StdLib for XC

```
cd $GOROOT
hg pull
hg update release

cd src
GOOS=linux ./all.bash
GOOS=linux GOARCH=386 ./all.bash
./all.bash
```

The Manual Way

```
GOOS=linux go build
```



COMMUNITY TOOLS

- https://github.com/mitchellh/gox
  - Builds multiple os/arch combinations in parallel
- https://github.com/laher/goxc
  - Designed for distribution:
    - Github.com releases: .zip files with documentation included
    - Debian .debs files
    - configurable build steps (testing, linking, etc)

I use both.

Getting & Installing

PACKAGE URLS

- go help importpath

```
import "fmt"
import "github.com/amattn/deeperror"
import "github.com/collectivehealth/tesl"
```

PACKAGE URLS

```
// Bitbucket (Git, Mercurial)

import “bitbucket.org/user/project"
import "bitbucket.org/user/project/sub/directory"

// GitHub (Git)

import "github.com/user/project"
import "github.com/user/project/sub/directory"

// Google Code Project Hosting (Git, Mercurial, Subversion)

import "code.google.com/p/project"
import "code.google.com/p/project/sub/directory"

import "code.google.com/p/project.subrepository"
import "code.google.com/p/project.subrepository/sub/directory"

// Launchpad (Bazaar)

import "launchpad.net/project"
import "launchpad.net/project/series"
import "launchpad.net/project/series/sub/directory"

import "launchpad.net/~user/project/branch"
import "launchpad.net/~user/project/branch/sub/directory"
```

GET VS INSTALL

- go get
  - fetches packages and installs them
  - -u flag will update (fetch latest) then install
- go install
  - builds, then puts packages in the pkg/ dir and executables in $GOBIN

Vetting & Linting

GO VET

- The vet tool is a configurable linter from the go team
- Go famously does not have warnings… some of what would be a warning from other compilers has been moved to the vet tool
- Documentation:
  - https://godoc.org/code.google.com/p/go.tools/cmd/vet

```
go vet somecode.go
```

Example Output

```
source.go:32: HandleMessage passes Lock by value: main.PassiveHandler contains
sync.WaitGroup contains sync.Mutex

source.go:110: range variable topic enclosed by function

source.go:245: wrong number of args for format in Errorf call: 1 needed but 4 args

source.go:178: struct field tag `json:—` not compatible with reflect.StructTag.Get

source.go:611: missing argument for Sprintf("%d"): format reads arg 2, have only 1 args

source.go:617: no formatting directive in Fatalf call

source.go:130: Println call ends with newline

source.go:166: possible formatting directive in Fatal call

source.go:141: unreachable code

exit status 1
```

GO VET

- This tool may produce false positives or false negatives
- Consider using flags to filter out the more questionable items if used in automated or continuous build systems

TESTING & BENCHMARKING

TESTING

- http://golang.org/pkg/testing/
- Basic template:

```
package main

import "testing"

func TestXxx(t *testing.T) {
    …
    if x == bad {
        t.Error(“expected x is good, but got”, x)
    }
    if y == 0 {
        t.Error(“expected non-zero y, got”, y)
    }
}
func TestYyy(t *testing.T) {
    t.Fatal(“This will always fail")
}
```

USE QUICKCHECK!

```
import “testing/quick"

func TestOddMultipleOfThree(t *testing.T) {
  f := func(x int) bool {
    y := OddMultipleOfThree(x)
    return y%2 == 1 && y%3 == 0
  }
  if err := quick.Check(f, nil); err != nil {
    t.Error(err)
  }
}
```

inject random ints

```
func TestString(t *testing.T) {
  f := func(s string) bool {
    err := ProcessString(s)
    return err == nil
  }
  if err := quick.Check(f, nil); err != nil {
    t.Error(err)
  }
}
```

inject random strings

TESTING MISC.

- The best reference or guide for go tests is the test files in the go standard library
- Go tests do not have SetUp or TearDown helpers. You have to write your own wrappers.
- More:
    - http://dave.cheney.net/2013/06/09/writing-table-driven-tests-in-go
    - https://github.com/bmizerany/assert

COVERAGE

- Test coverage tool built into go as of version 1.2
- Install:

```
go get code.google.com/p/go.tools/cmd/cover
```

- In the simple case, will just spit out a %

```
$ go test -cover
PASS
coverage: 42.9% of statements
ok      size    0.026s
```

COVERAGE BY FUNCTION

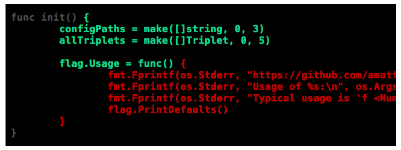
- Getting fancy, we check coverage by function:

```
$ go test -coverprofile=coverage.out
$ go tool cover -func=coverage.out

github.com/amattn/f/config.go:21:init42.9%
github.com/amattn/f/config.go:43:prependConfigPath0.0%
github.com/amattn/f/config.go:53:appendConfigPath66.7%
github.com/amattn/f/config.go:60:joinFrcToDir100.0%
github.com/amattn/f/config.go:205:cleanLine83.3%
github.com/amattn/f/main.go:15:init100.0%
github.com/amattn/f/main.go:23:main0.0%
github.com/amattn/f/triplet.go:20:IsValid0.0%
github.com/amattn/f/triplet.go:24:IsEqual66.7%
<SNIP>
github.com/amattn/f/triplet.go:89:PrintMenu0.0%
total:(statements)21.1%
```

Coverage Browser Output

- Getting extra fancy, generate color coded html output:

```
$ go test -coverprofile=coverage.out
$ go tool cover -html=coverage.out
```

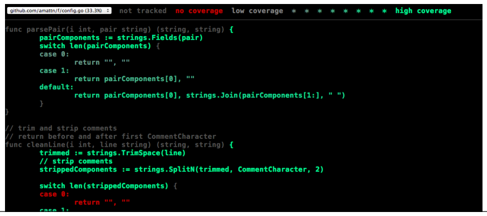


```
func init() {
        configPaths = make([]string, 0, 3)
        allTriplets = make([]Triplet, 0, 5)

        flag.Usage = func() {
                fmt.Fprintf(os.Stderr, "https://github.com/amatt
                fmt.Fprintf(os.Stderr, "Usage of %s:\n", os.Args
                fmt.Fprintf(os.Stderr, "Typical usage is 'f <Num
                flag.PrintDefaults()
        }
}
```

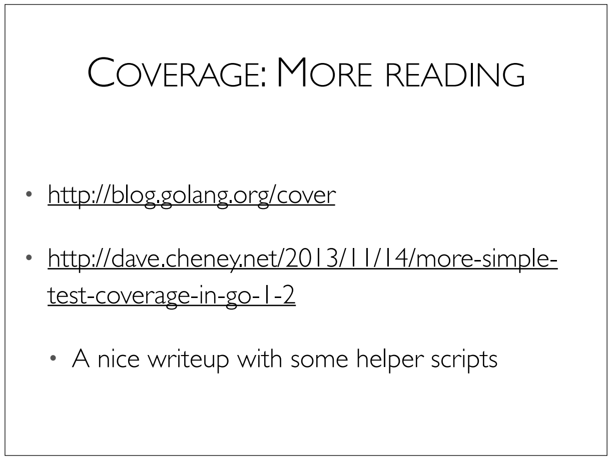
COVERAGE HEATMAP OUTPUT

- Getting extra, extra fancy, generate heat maps:

```
$ go test -covermode=count -coverprofile=count.out
$ go tool cover -html=count.out
```



```
func parsePair(i int, pair string) (string, string) {
        pairComponents := strings.Fields(pair)
        switch len(pairComponents) {
        case 0:
                return "", ""
        case 1:
                return pairComponents[0], ""
        default:
                return pairComponents[0], strings.Join(pairComponents[1:], " ")
        }
}

// trim and strip comments
// return before and after first CommentCharacter
func cleanLine(i int, line string) (string, string) {
        trimmed := strings.TrimSpace(line)
        // strip comments
        strippedComponents := strings.SplitN(trimmed, CommentCharacter, 2)

        switch len(strippedComponents) {
        case 0:
                return "", ""
        case 1:
```


COVERAGE: MORE READING

- http://blog.golang.org/cover
- http://dave.cheney.net/2013/11/14/more-simple-test-coverage-in-go-1-2
  - A nice writeup with some helper scripts

If it's not obvious by now, Dave Cheney's blog is a must read/ subscribe for all gophers

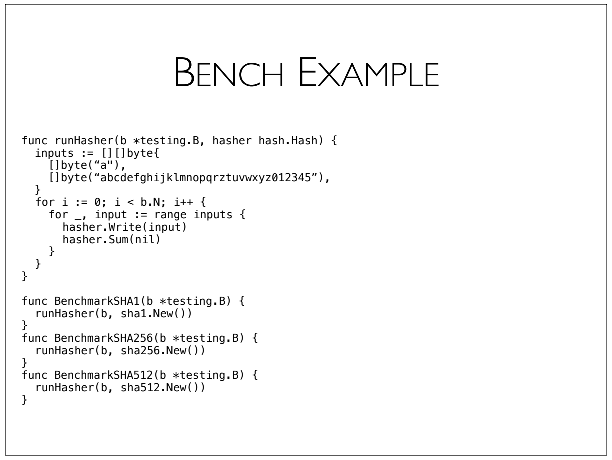
BENCHMARKING

- This should look familiar:

```
package main

import "testing"

func BenchXxx(b *testing.B) {

}
func BenchYyy(b *testing.B) {

}
```

BENCHMARKING

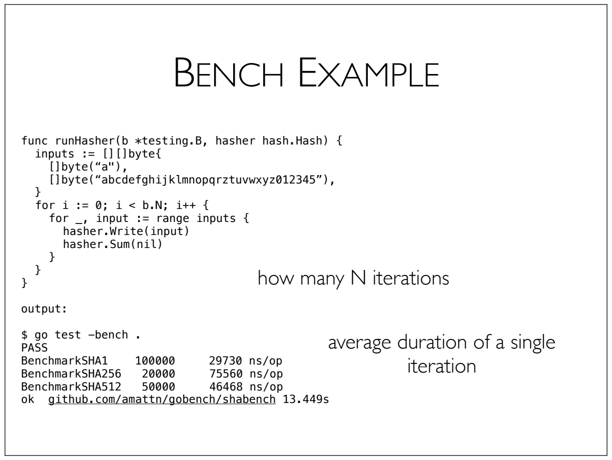
```
package main

import "testing"

func BenchXxx(b *testing.B) {
    for i := 0; i < b.N; i++ {
        fmt.Sprintf("hello")
    }
}
```



BENCH EXAMPLE

```
func runHasher(b *testing.B, hasher hash.Hash) {
  inputs := [][]byte{
    []byte("a"),
    []byte("abcdefghijklmnopqrztuvwxyz012345"),
  }
  for i := 0; i < b.N; i++ {
    for _, input := range inputs {
      hasher.Write(input)
      hasher.Sum(nil)
    }
  }
}

func BenchmarkSHA1(b *testing.B) {
  runHasher(b, sha1.New())
}
func BenchmarkSHA256(b *testing.B) {
  runHasher(b, sha256.New())
}
func BenchmarkSHA512(b *testing.B) {
  runHasher(b, sha512.New())
}
```

https://github.com/amattn/gobench



Bench Example

```
func runHasher(b *testing.B, hasher hash.Hash) {
  inputs := [][]byte{
    []byte(“a"),
    []byte(“abcdefghijklmnopqrztuvwxyz012345”),
  }
  for i := 0; i < b.N; i++ {
    for _, input := range inputs {
      hasher.Write(input)
      hasher.Sum(nil)
    }
  }
}
```

how many N iterations

```
output:

$ go test -bench .
PASS
BenchmarkSHA1    100000     29730 ns/op
BenchmarkSHA256   20000     75560 ns/op
BenchmarkSHA512   50000     46468 ns/op
ok  github.com/amattn/gobench/shabench 13.449s
```

average duration of a single iteration

https://github.com/amattn/gobench

PROFILING

PROFILING AN EXCUTABLE

- use the pprof package

```
import “runtime/pprof”

var cpuprofile = flag.String("cpuprofile", "", "write cpu profile to file")

func main() {
    flag.Parse()
    if *cpuprofile != "" {
        f, err := os.Create(*cpuprofile)
        defer f.Close()
        if err != nil {
            log.Fatal(err)
        }
        pprof.StartCPUProfile(f)
        defer pprof.StopCPUProfile()
    }
    ...
```

Profiling an Excutable

generate cpu.prof

```
go build

go build && ./shabench -cpuprofile=cpu.prof

go tool pprof shabench cpu.prof
```

launch pprof with the generated profiles

CPU EXAMPLE OUTPUT

- top is the most useful command:

```
$ go tool pprof shabench cpu.prof
Welcome to pprof!  For help, type 'help'.
(pprof) top
Total: 1414 samples
    1413  99.9%  99.9%     1413  99.9%
runtime.mach_semaphore_timedwait
       1   0.1% 100.0%        1   0.1% runtime.mach_semaphore_signal
       0   0.0% 100.0%        1   0.1% crypto/sha1.(*digest).Sum
       0   0.0% 100.0%        1   0.1% growslice1
       0   0.0% 100.0%        1   0.1% main.doOneHash
       0   0.0% 100.0%        1   0.1% main.main
       0   0.0% 100.0%     1413  99.9% notetsleep
       0   0.0% 100.0%     1413  99.9% runtime.MHeap_Scavenger
       0   0.0% 100.0%        1   0.1% runtime.gc
       0   0.0% 100.0%     1414 100.0% runtime.gosched0
```

BENCH HAS PROFILING BUILT-IN

- http://golang.org/cmd/go/#hdr-Description_of_testing_flags

```
go test --cpuprofile cpu.prof -bench .
```

Write a CPU profile to the specified file before exiting.

```
go test --memprofile mem.prof -bench .
```

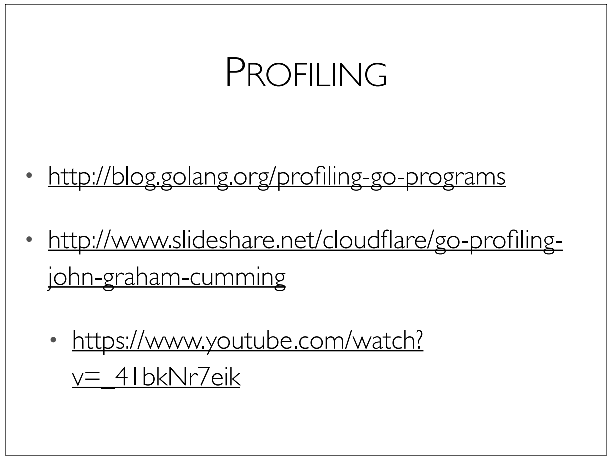
Write a memory profile to the specified file after all tests have passed.

BENCH PPROF

generate shabench.test and mem.prof

```
cd github.com/amattn/gobench/shabench

go test --memprofile mem.prof -bench .

go tool pprof shabench.test mem.prof
```

launch pprof with the generated profiles



Profiling

- http://blog.golang.org/profiling-go-programs
- http://www.slideshare.net/cloudflare/go-profiling-john-graham-cumming
  - https://www.youtube.com/watch?v=_41bkNr7eik

Also Dave Cheney:

http://dave.cheney.net/2013/07/07/introducing-profile-super-simple-profiling-for-go-programs

DEBUGGING

- https://golang.org/doc/gdb
- http://blog.golang.org/race-detector

Dependency Management

Go Dependency Management

- go get not quite sufficient
- As of Summer 2014, >25 different tools related to go dependency management
- A dedicated google group for the topic:
  - https://groups.google.com/forum/?hl=en-GB#!forum/go-package-management

Some good writing on the topic

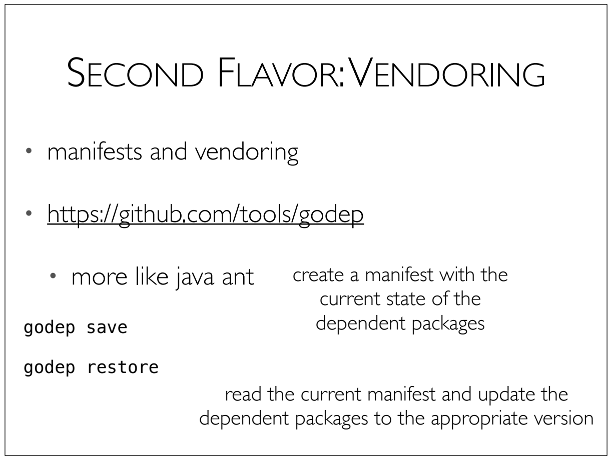
- http://nathany.com/go-packages/
- http://dave.cheney.net/2014/03/22/thoughts-on-go-package-management-six-months-on
- https://docs.google.com/document/d/1k-3mwBqAdTIKGciIWZPuKSMy3DWtfNRFDs9o98lcwHY/

First Flavor: Redirection

- package url versioning
- https://gopkg.in/

```
go get gopkg.in/yaml.v1
```

gopkg.in essentially redirects to the v1 tag of the yaml package



Second Flavor: Vendoring

- manifests and vendoring
- https://github.com/tools/godep
  - more like java ant

create a manifest with the current state of the dependent packages

```
godep save
godep restore
```

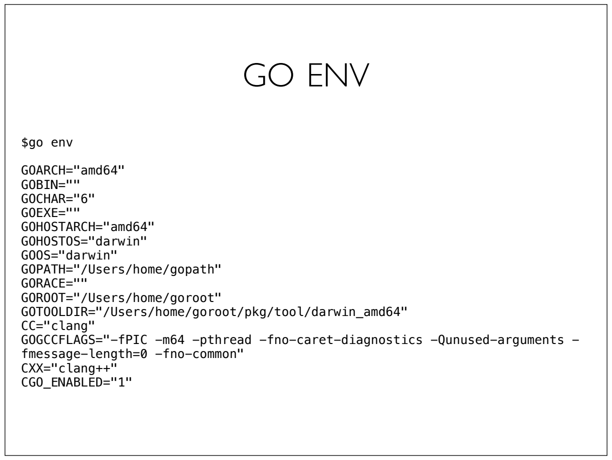
read the current manifest and update the dependent packages to the appropriate version

FWIW, I prefer this

Misc

LEFTOVERS

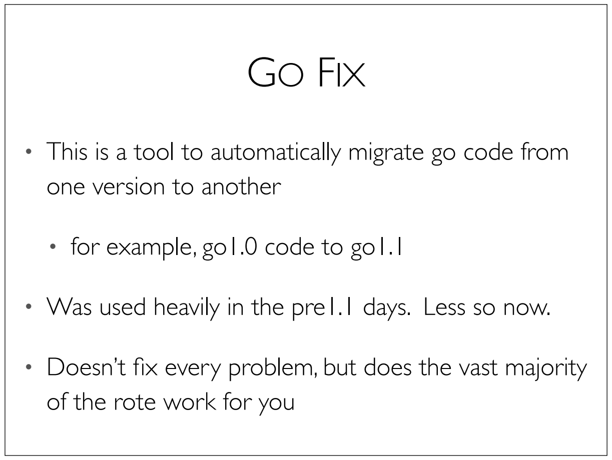
- go version
  - `go version go1.3.1 darwin/amd64`
- go env
- go list
- go fix
- go present
- http://godoc.org/code.google.com/p/go.tools/cmd/oracle



GO ENV

```
$go env

GOARCH="amd64"
GOBIN=""
GOCHAR="6"
GOEXE=""
GOHOSTARCH="amd64"
GOHOSTOS="darwin"
GOOS="darwin"
GOPATH="/Users/home/gopath"
GORACE=""
GOROOT="/Users/home/goroot"
GOTOOLDIR="/Users/home/goroot/pkg/tool/darwin_amd64"
CC="clang"
GOGCCFLAGS="-fPIC -m64 -pthread -fno-caret-diagnostics -Qunused-arguments -
fmessage-length=0 -fno-common"
CXX="clang++"
CGO_ENABLED="1"
```

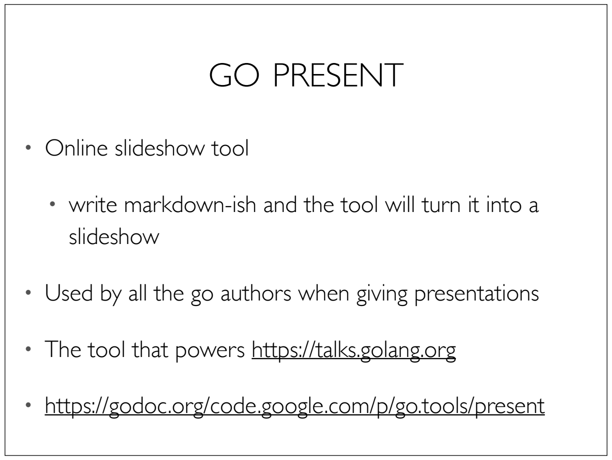
Super useful for debugging someone else’s environment



Go Fix

- This is a tool to automatically migrate go code from one version to another
  - for example, go1.0 code to go1.1
- Was used heavily in the pre1.1 days. Less so now.
- Doesn't fix every problem, but does the vast majority of the rote work for you

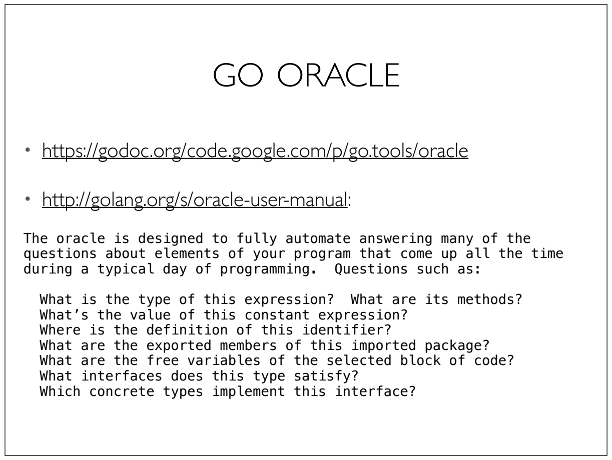
It's hard to underestimate the impact this tool had on the quality of the language. Because the authors had this tool in the pre-1.0 days, they weren't afraid to make backward incompatible, but better long-term changes early on. They could go fix the vast majority of them and the end result is that we have a vastly improved language and std lib because of it.



GO PRESENT

- Online slideshow tool
  - write markdown-ish and the tool will turn it into a slideshow
- Used by all the go authors when giving presentations
- The tool that powers https://talks.golang.org
- https://godoc.org/code.google.com/p/go.tools/present

I don’t use it. I like keynote. Get off my lawn.


GO ORACLE

- https://godoc.org/code.google.com/p/go.tools/oracle
- http://golang.org/s/oracle-user-manual:

```
The oracle is designed to fully automate answering many of the
questions about elements of your program that come up all the time
during a typical day of programming.  Questions such as:

  What is the type of this expression?  What are its methods?
  What’s the value of this constant expression?
  Where is the definition of this identifier?
  What are the exported members of this imported package?
  What are the free variables of the selected block of code?
  What interfaces does this type satisfy?
  Which concrete types implement this interface?
```

This this is just cool.

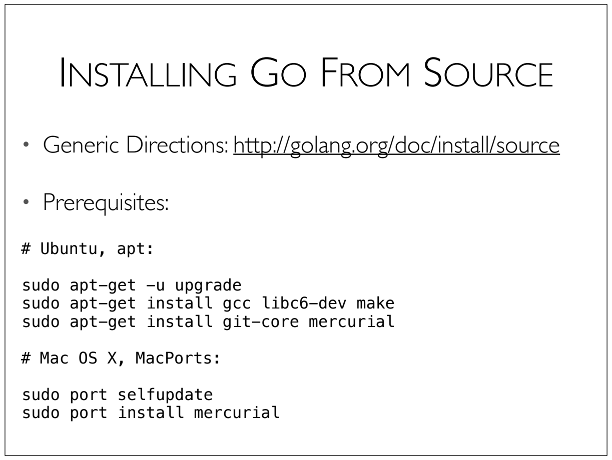
Not really useful unless integrated into your IDE however.

Generation

Go generate: A Proposal

- A proposal for a tool to help with automated tasks in go code
- Under proposal for Go 1.4
- Designed to replace make for the subset of uses that typical go package maintainers use make for
  - generate protobufs, yacc, bindata, embedding html and other markup in code
- https://groups.google.com/forum/#!topic/golang-dev/ZTD1qtpruA8

Installing & Updating

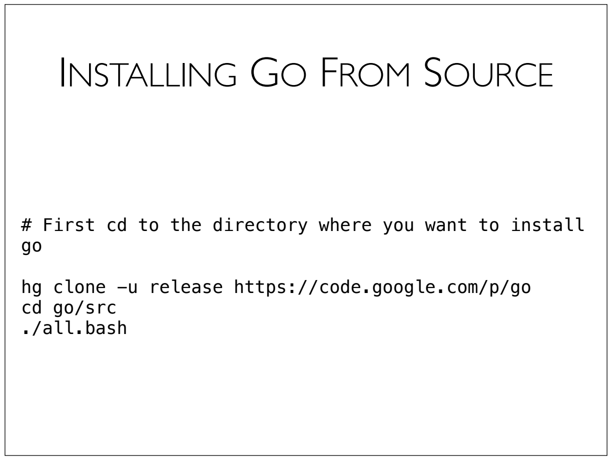


Installing Go From Source

- Generic Directions: http://golang.org/doc/install/source
- Prerequisites:

```
# Ubuntu, apt:

sudo apt-get -u upgrade
sudo apt-get install gcc libc6-dev make
sudo apt-get install git-core mercurial

# Mac OS X, MacPorts:

sudo port selfupdate
sudo port install mercurial
```

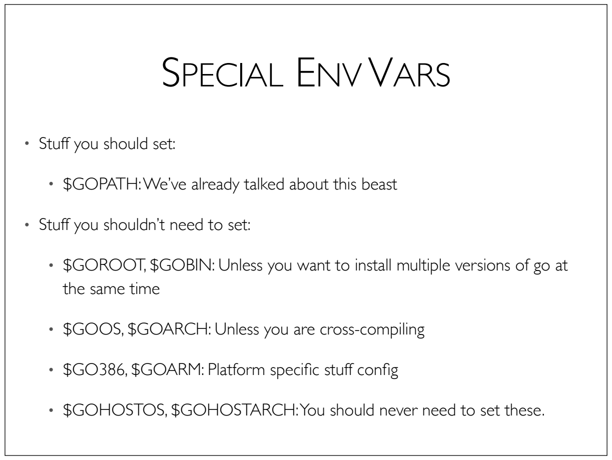
if you compile/test on a darwin system and deploy to linux, it's usually easier to install via source manually for cross compiling.



INSTALLING GO FROM SOURCE

```
# First cd to the directory where you want to install
go

hg clone -u release https://code.google.com/p/go
cd go/src
./all.bash
```

if you compile/test on a darwin system and deploy to linux, it’s usually easier to install via source manually for cross compiling.



SPECIAL ENV VARS

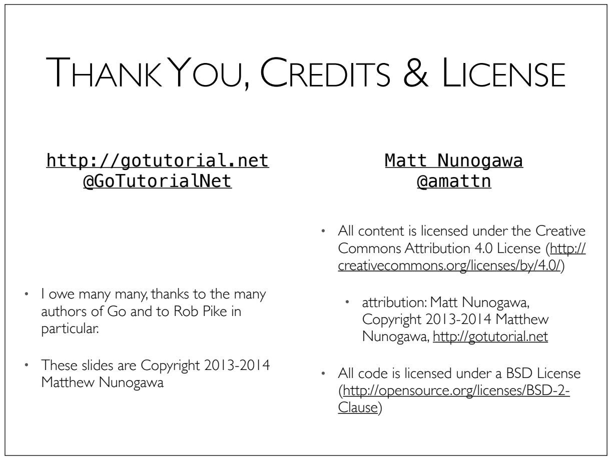
- Stuff you should set:
  - $GOPATH: We've already talked about this beast
- Stuff you shouldn't need to set:
  - $GOROOT, $GOBIN: Unless you want to install multiple versions of go at the same time
  - $GOOS, $GOARCH: Unless you are cross-compiling
  - $GO386, $GOARM: Platform specific stuff config
  - $GOHOSTOS, $GOHOSTARCH: You should never need to set these.

http://golang.org/doc/install/source#environment

http://dave.cheney.net/2013/06/14/you-dont-need-to-set-goroot-really

Updating Go

```
cd $GOROOT
hg pull
hg update release

cd src
./all.bash
```



Thank You, Credits & License

http://gotutorial.net
@GoTutorialNet

Matt Nunogawa
@amattn

- I owe many many, thanks to the many authors of Go and to Rob Pike in particular.
- These slides are Copyright 2013-2014 Matthew Nunogawa

These are the slides that I used to learn go back in 2011.
“out of date”: The actually syntax has not significantly changed.
Some of the terminology is no longer in use, typically because after contact with the community, misunderstandings have occurred.

In the creation of these slides, I have, to the utmost of my ability, attempted to make sure that these are correct and updated. Any errors are likely my fault. I make no guarantee that these slides are correct or will remain correct under the inevitable progression of time.