Go for DevOps

Linux.conf.au 2017 Sysadmin Miniconf

Caskey L. Dickson

caskey@{microsoft,gmail,twitter,github, ...}

@caskey

What’s go?

Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.

```
package main
import "fmt"

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

@caskey

Platform wars are over

(Hint: we all won)

What’s go?

Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.

```
package main
import "fmt"

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

@caskey

Building for the current system (Linux 64 bit)

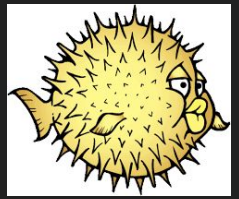
```
$ go build main.go

$ file main

main: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
statically linked, not stripped
```

Go is purrrrfect for devops

@caskey

Explosion of platforms




Building for windows (64 bit)

```
$ GOOS=windows go build main.go

$ file main.exe

main.exe: PE32+ executable (console) x86-64 (stripped to
external PDB), for MS Windows
```

@caskey

Building for OSX

```
$ GOOS=darwin go build main.go

$ file main

main: Mach-O 64-bit x86_64 executable
```

@caskey

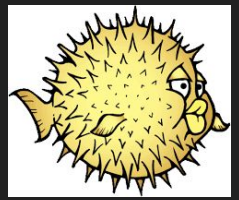
Building for NetBSD

```
$ GOOS=netbsd go build main.go

$ file main

main: ELF 64-bit LSB executable, x86-64, version 1 (NetBSD),
statically linked, for NetBSD 5.99, not stripped
```

@caskey

Building for OpenBSD

```
$ GOOS=openbsd go build main.go

$ file main

main: ELF 64-bit LSB executable, x86-64, version 1
(OpenBSD), statically linked, for OpenBSD, not stripped
```



Building for FreeBSD

```
$ GOOS=freebsd go build main.go

$ file main

main: ELF 64-bit LSB executable, x86-64, version 1
(FreeBSD), statically linked, not stripped
```

Building for Solaris

```
$ GOOS=solaris go build main.go

$ file main

main: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib/amd64/ld.so.1, not
stripped
```

@caskey

Building for windows (32 bit)

```
$ GOOS=windows GOARCH=386 go build main.go

$ file main.exe

main.exe: PE32 executable (console) Intel 80386 (stripped to
external PDB), for MS Windows
```

@caskey

Bash is great,
Python is great,
but Go is better.

Microservices are services

Microservices

Microservices

Micro Services

Micro **SERVICES**

@caskey

No small systems

It’s never “**just a shell script**”

Ops tools are not 2nd class citizens, they deserve a proper engineering lifecycle

“Real” programming languages have features that make life easier

- Type safety
- Classes
- Packages
- Release management

DevOps

Congratulations, you’re all software engineers now.

Sysops pushing buttons and running scripts is dying.

Good.

Job control

```
package main

import (
     "fmt"
     "log"
     "os/exec"
)

func main() {
     out, err := exec.Command("date").Output()
     if err != nil {
          log.Fatal(err)
     }
     fmt.Printf("The date is %s\n", out)
}
```

@caskey

Platform specific code

main.go:

```
out, err := exec.Command("ifconfig", "-a").Output()
```

@caskey

Refactor the command line part

main.go:

```
~~out, err := exec.Command("ifconfig", "-a").Output()~~

cmd := GetCommand()
out, err := exec.Command(cmd[0], cmd[1:]...).Output()
```

Add a helper that generates the args

main.go:

```
func GetCommand() []string {
     return []string{“ifconfig”, “-a”}
}

cmd := GetCommand()
out, err := exec.Command(cmd[0], cmd[1:]...).Output()
```

Platform specific code

main_linux.go:

```
func GetCommand() []string {
     return []string{“ifconfig”, “-a”}
}
```

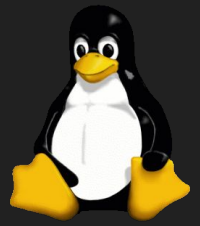
main.go:

```
cmd := GetCommand()
out, err := exec.Command(cmd[0], cmd[1:]...).Output()
```

Platform specific code

main_linux.go:

```
func GetCommand() []string {
    return []string{“ifconfig”, “-a”}
}
```



main_windows.go:

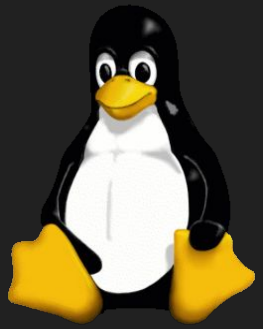
```
func GetCommand() []string {
    return []string{“ipconfig”, “/all”}
}
```

main.go:

```
cmd := GetCommand()
out, err := exec.Command(cmd[0], cmd[1:]...).Output()
```

@caskey

What about this guy?



Platform specific code

main_linux.go:

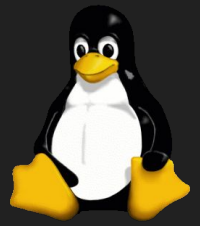
```
func GetCommand() []string {
     return []string{“ifconfig”, “-a”}
}
```

main_windows.go:

```
func GetCommand() []string {
     return []string{“ipconfig”, “/all”}
}
```

main_freebsd.go:

```
func GetCommand() []string {
     return []string{“ifconfig”, “-a”}
}
```

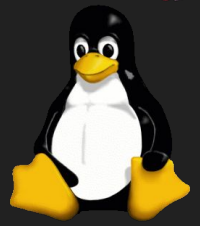


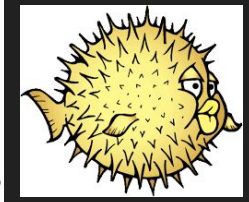
main.go:

```
cmd := GetCommand()
out, err := exec.Command(cmd[0], cmd[1:]...).Output()
```

@caskey

Platform specific code





main_unixlike.go:

```
// +build linux darwin netbsd openbsd freebsd

func GetCommand() []string {
     return []string{"ifconfig", "-a"}
}
```

main_windows.go:

```
func GetCommand() []string {
     return []string{"ipconfig", "/all"}
}
```

main.go:

```
cmd := GetCommand()
out, err := exec.Command(cmd[0], cmd[1:]...).Output()
```

@caskey

Two mechanisms

Naming the file with the build restriction

```
foo_${GOOS}.go
```

Adding build directives to the top of the file

```
// +build linux,386 darwin
```

@caskey

Specify target operating system

$GOOS environment variable for operating system

```
GOOS=windows go build ...
```

$GOARCH environment variable for cpu architecture

```
GOARCH=386 go build ...
```

@caskey

Lots of supported targets

android	arm
darwin	386
darwin	amd64
darwin	arm
darwin	arm64
dragonfly	amd64
freebsd	386
freebsd	amd64
freebsd	arm
linux	386
linux	amd64
linux	arm
linux	arm64
linux	ppc64
linux	ppc64le
linux	mips64
linux	mips64le
netbsd	386
netbsd	amd64
netbsd	arm
openbsd	386
openbsd	amd64
openbsd	arm
plan9	386
plan9	amd64
solaris	amd64
windows	386
windows	amd64

@caskey

You’re now writing services, not scripts

Services need the full software development lifecycle

Golang has features that make the SDLC better (and cross platform)

Questions?

@caskey