

# BUILDING AN AWESOME CLI APP IN GO

NEW AND IMPROVED



# ASHLEY MAC

- Dev Advocate at Rackspace
- Go newbie
- The funny one

# SPF13

- Go Team at Google
- Product/UX focused engineer
- The good looking one

**FOLLOW US ON  
TWITTER  
@ASHLEYMCNAMARA  
@SPF13**

# AGENDA

- Introduction to UX guidelines for CLIs
- Design our experience
- Light introduction to Go
- Build application

# GROUND RULES

- This is an interactive workshop
- Your participation is needed
- Will follow the 50% rule

GO  
ENVIRONMENT  
CHECK



~  
» |

**OPEN A  
TERMINAL**

# GO ENV

> go env

GOBIN="/Users/<youruser>/go/bin"

GOPATH="/Users/<youruser>/go"

# GO ENV

› go env

GOBIN="/Users/sfrancia/go/bin"

GOPATH="/Users/sfrancia/go"

*IF THESE AREN'T SET STOP  
NOW AND RAISE YOUR HAND*

# GOPATH

- The Go toolset uses an environment variable called GOPATH to find & store Go source code.
- You can set GOPATH to anything you want, but things will be easier if it's set in your home directory.

# GO PATH

**Windows:**

```
c:\ setx GOPATH %USERPROFILE%\go
```

**OS X:**

```
> echo 'export GOPATH=$HOME/go\n' >> ~/.bash_profile
```

**Close the terminal, reopen it, and type the following:**

```
> echo $GOPATH
```

# GET & INSTALL COBRA

```
> go get -u \
```

```
github.com/spf13cobracobra
```

# COBRA

› cobra

Cobra is a Cli library for Go that empowers applications. This application is a tool to generate the needed files to quickly create a Cobra application.

Usage:

cobra [command]

Available Commands:

add            Add a command to a Cobra Application

# COBRA

› cobra

Cobra is a cli library for Go that empowers applications. This application is a tool to generate the needed files to quickly create a Cobra application.

Usage:

cobra [command]

**IF COBRA DOESN'T WORK**

Available Commands:

add

**RAISE YOUR HAND**

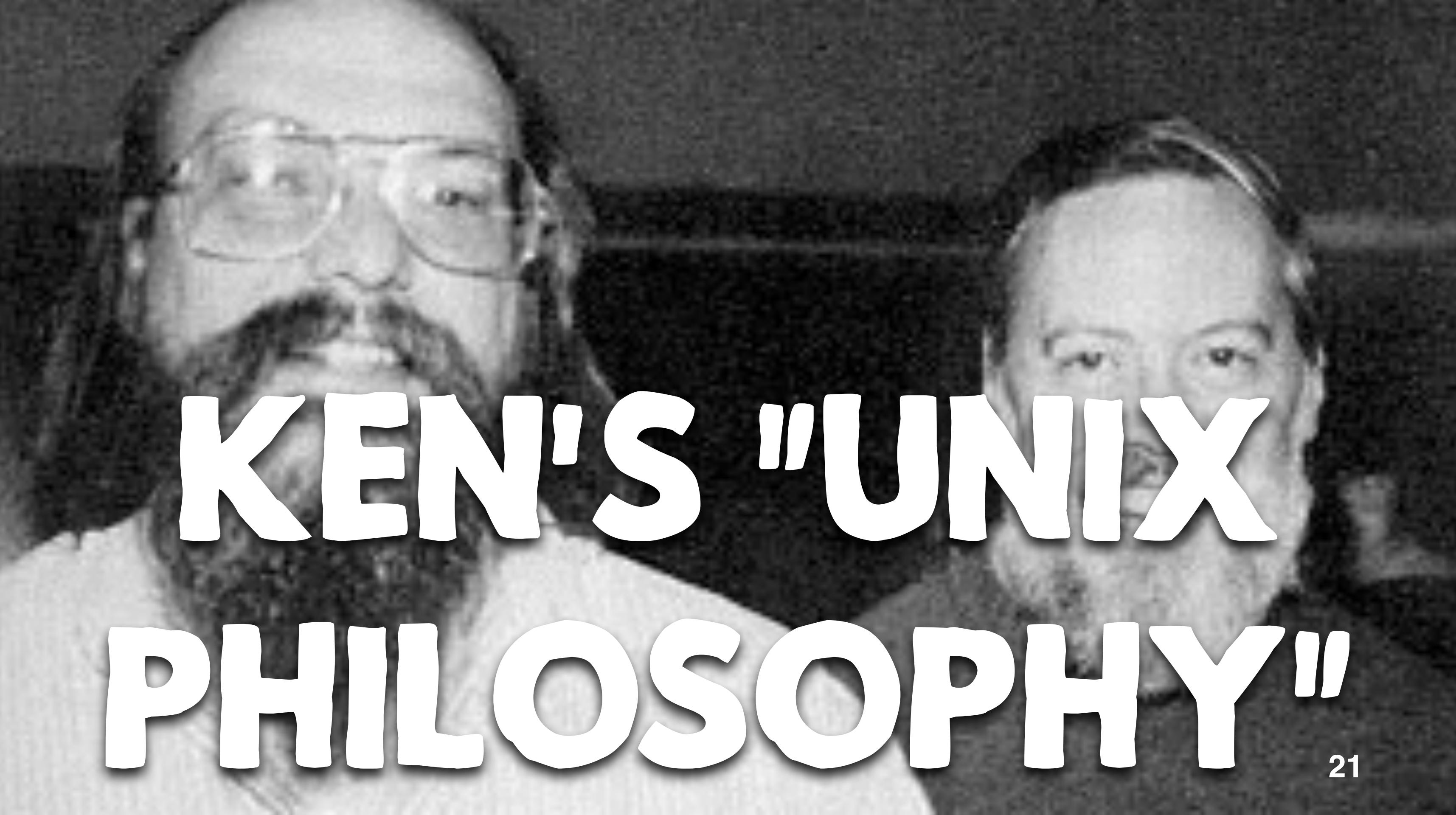
Add a command to a Cobra Application

SPF13.COM/  
OSCON.PDF

UX OF  
cli

# HUMAN INTERFACE GUIDELINES FOR CLIS

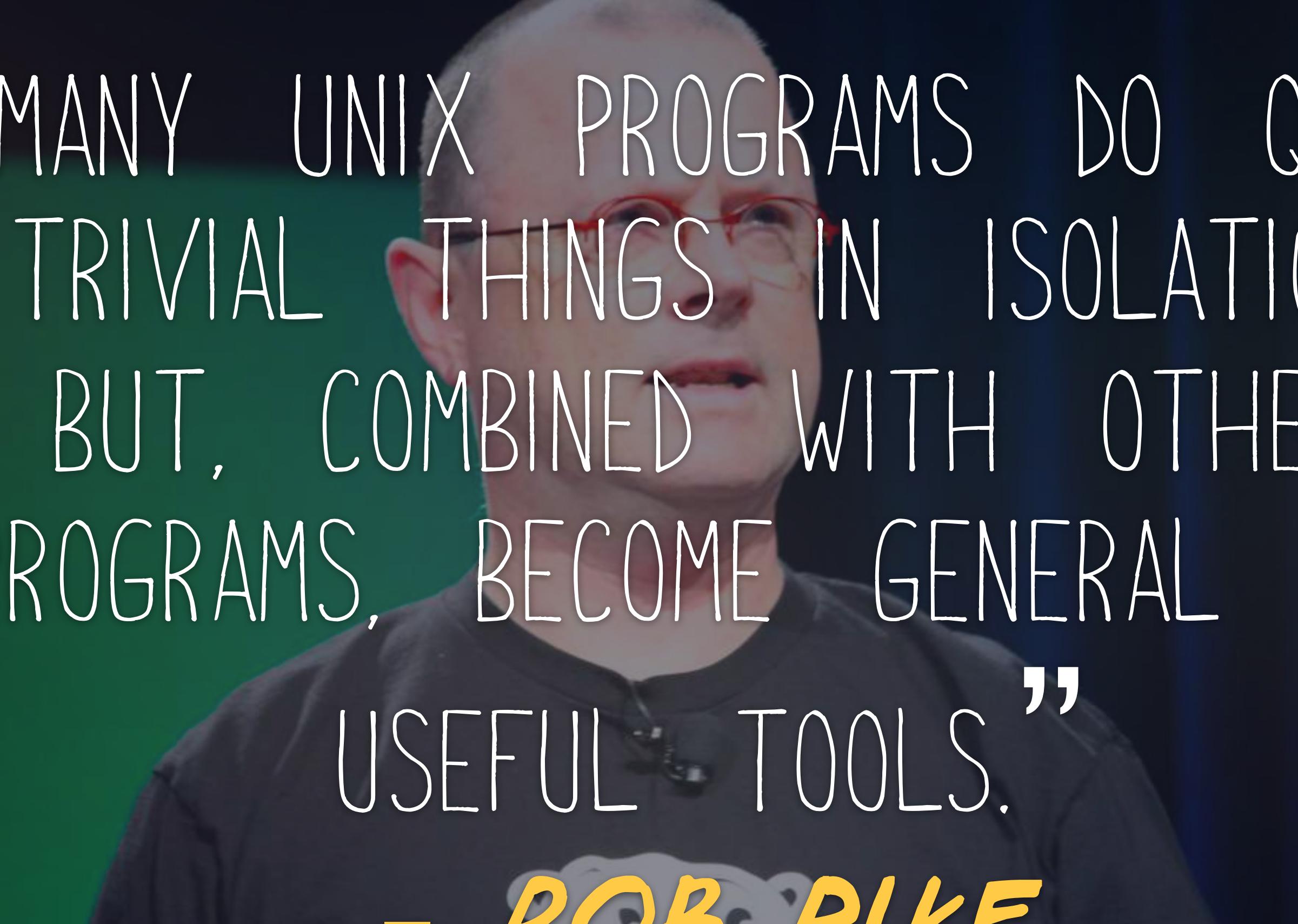
# UNIX PHILOSOPHY



**KEN'S "UNIX  
PHILOSOPHY!"**

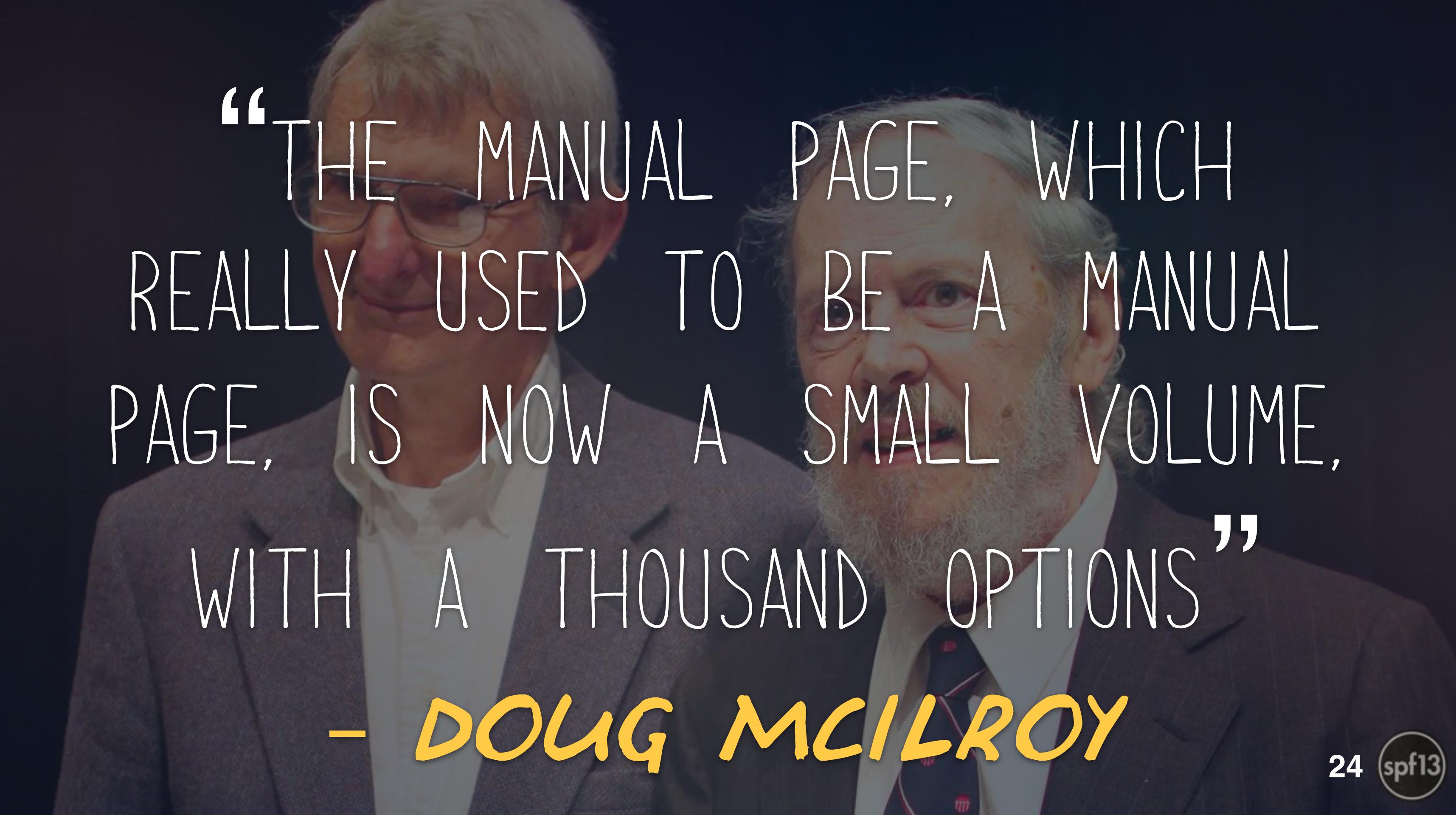
# UNIX PHILOSOPHY

- Simple
- Clear
- Composable
- Extensible
- Modular
- Small



“MANY UNIX PROGRAMS DO QUITE TRIVIAL THINGS IN ISOLATION, BUT, COMBINED WITH OTHER PROGRAMS, BECOME GENERAL AND USEFUL TOOLS.”

- ROB PIKE



“THE MANUAL PAGE, WHICH  
REALLY USED TO BE A MANUAL  
PAGE, IS NOW A SMALL VOLUME,  
WITH A THOUSAND OPTIONS”

- DOUG MCILROY

**POSIX +**  
**GNU**

# COMMANDS

# WHAT ARE COMMANDS



# COMMANDS

> ls

c:\dir

# ABBREVIATED

ls

# list

cp

# copy

# > cat

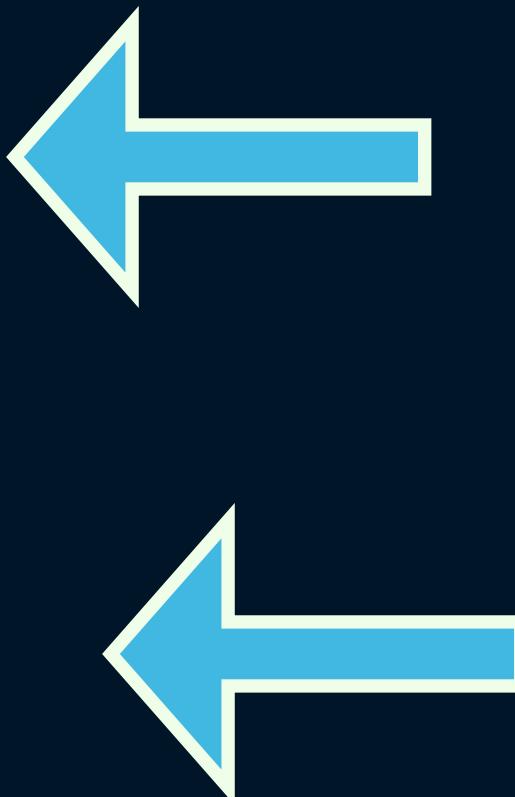
# concatenate

# cd

# change directory

# SHORT

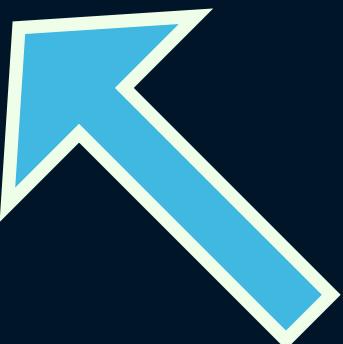
- > ls
- > cp
- > cat
- > cd



*SHORT  
&  
CLEAR*

# CONTEXT

> ls



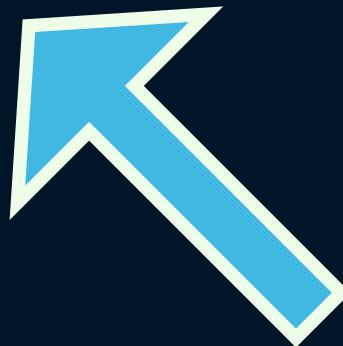
*OPERATES IN CURRENT  
DIRECTORY*

A COMMAND  
DOES  
SOMETHING

# THE LANGUAGE OF COMMANDS

# HAS A LANGUAGE

> ls



*LIST THE CONTENTS  
OF THE DIRECTORY I'M IN*

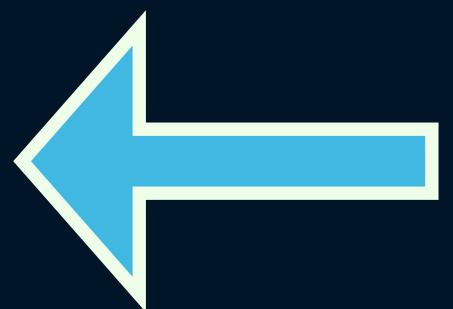
# COMMANDS

> ls

> rm

> zip

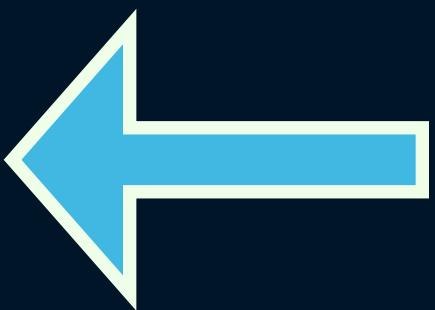
> find



*VERBS*

# COMMANDS

c:\dir



NOUN?

# ARGS

WHAT ARE  
ARGS?

# INPUT

› rm [file]

› cp [file] [newfile]

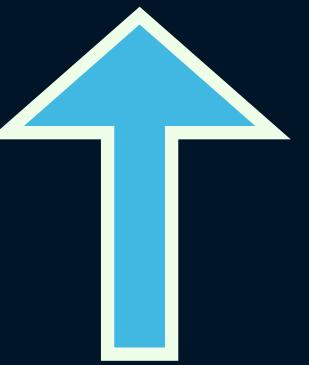
c:\copy [file] [newfile]

# MANY INPUTS

```
> rm [file] ... [fileN]
```



ARG0



ARGN

# ORDER MATTERS

```
> cp [file] [newfile]
```

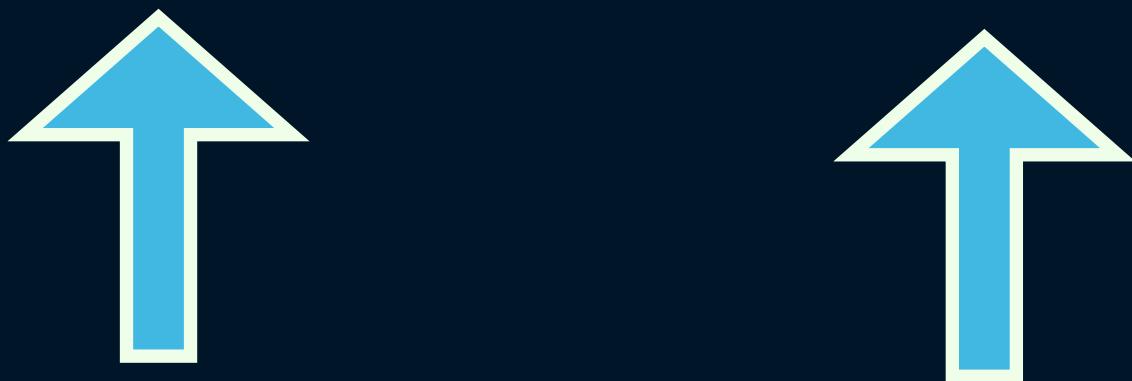
The diagram illustrates the cp command with two blue arrows pointing upwards. The first arrow points from the word 'FROM' to the first bracketed placeholder '[file]'. The second arrow points from the word 'TO' to the second bracketed placeholder '[newfile]'. The word 'FROM' is positioned below the first arrow, and the word 'TO' is positioned below the second arrow.

FROM

TO

# SEPARATED

```
> cp [file] [newfile]
```



*SPACE SEPARATES*

AN ARGUMENT  
IS  
SOMETHING

# THE LANGUAGE OF ARGS

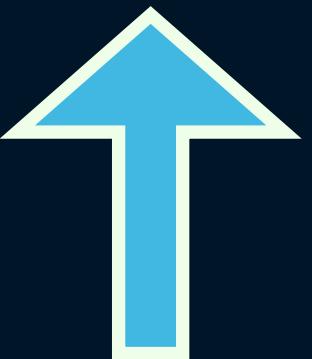
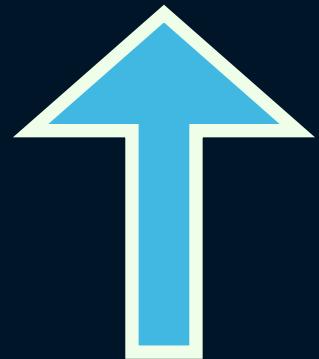
# DIRECT OBJECT

> cp [file] [newfile]

VERB      NOUN      NOUN

# IS PROOUNCABLE

```
> cp [file] [newfile]
```



*COPY THIS FILE TO HERE*

# OPTIONS

# FLAGS

# MODIFY ACTIONS

› rm [options] [file]

c:\del [options] [file]

# SEPARATORS

```
› rm -f badfile.txt
```



*SPACE SEPARATES*

# PREFIXED

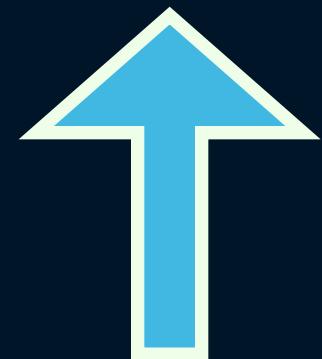
```
› rm --force
```



*PREFIX*

# PREFIXED

```
> rm -f
```



*PREFIX*

# PREFIXED

c:\ del /F



*PREFIX*

# SHORT VS LONG

-f == --force

COMMON OPTIONS  
SHORTENED

# STACKABLE

```
› rm -r -f → rm -rf
```

*SHORT OPTIONS STACK*

A FLAG  
MODIFIES  
BEHAVIOR

# THE LANGUAGE OF FLAGS

# ADVERB

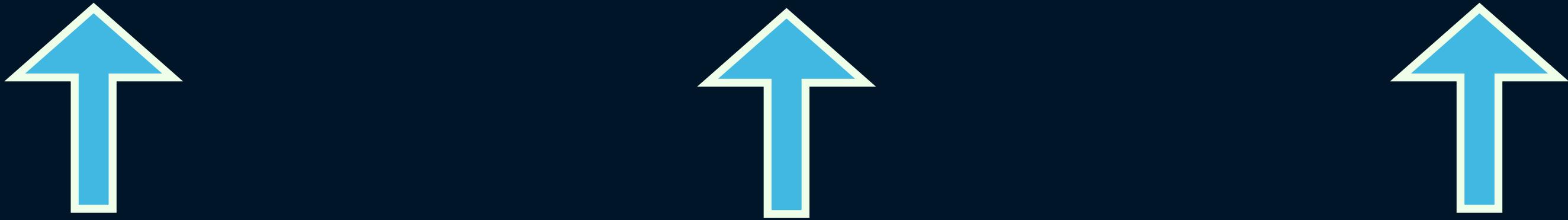
```
> rm --force [file]
```

↑      ↑      ↑

**VERB**    **ADVERB**    **NOUN**

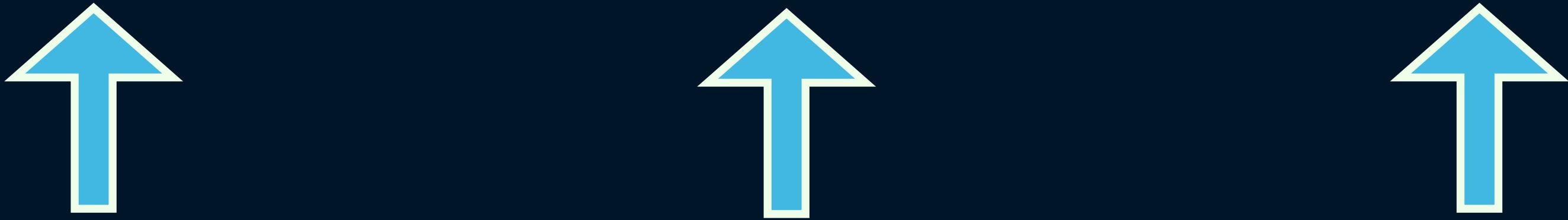
# PRONOUNCEABLE

```
› rm --force [file]
```



# PRONOUNCEABLE

```
> rm --force [file]
```



*REMOVE THIS FILE  
WITH FORCE*

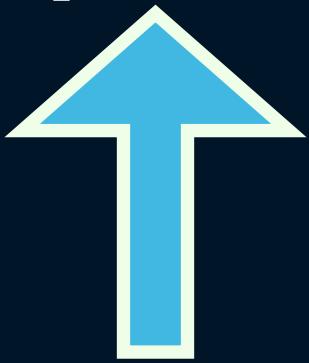
# FLAG INPUT

```
> ls --color /home/spf13
```

VERB                    ADVERB                    NOUN

# PRONOUNCEABLE

```
> ls --color /home/spf13
```



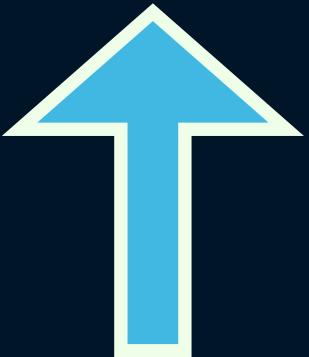
COLORFULLY LIST MY  
HOME DIRECTORY

# MODIFY BEHAVIOR

```
> ls -a
```

# MODIFY BEHAVIOR

```
> ls -a
```



*LIST ALL THE THINGS*

ALL ISN'T  
AN ADVERB

# ADVERBS EXPRESS

- manner
- place
- time
- frequency
- degree
- level of certainty
- etc.

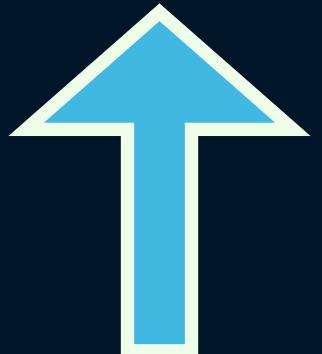
# ADVERBS ANSWER

- How?
- In what way?
- When?
- Where?
- To what extent?

# MODIFY BEHAVIOR

to what extent?

```
> ls -a
```

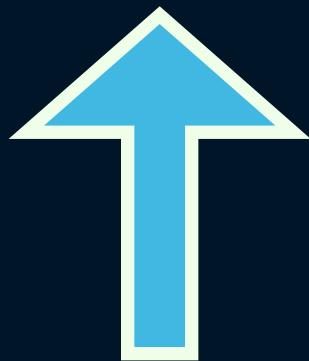


*LIST COMPLETELY*

# INPUT-ABLE

to what extent?

```
> ls --width=40
```



*INPUT*

# INPUT-ABLE

to what extent?

```
> ls --width 40
```



*INPUT*

# PREPOSITION

>

ls

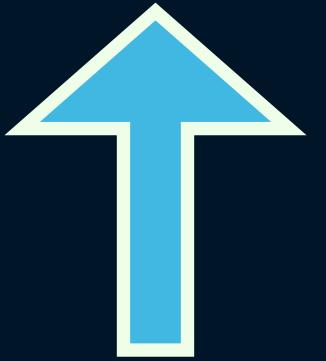
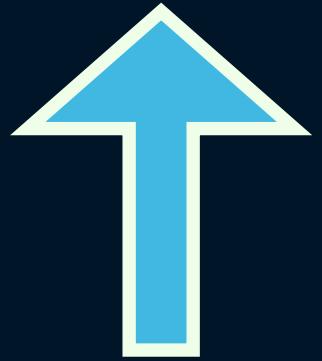
--width

40

*VERB*

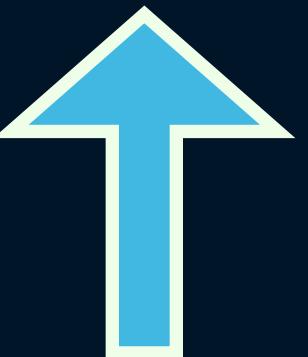
*PREPOSITION*

*OBJ*



# PRONOUNCEABLE

```
> ls --width 40
```



*LIST THE DIRECTORY WITH  
A WIDTH OF 40 COLS*

# PREP PHRASE = ADVERB

- consists of a preposition and its object
- acts as an adverb
- "Speaking at OSCON"

# BAD FLAG DESIGN

# FLAGS AS ACTIONS

# FLAG = ACTION

```
> tar -xvf
```

# FLAG = ACTION

› tar

First option must be a mode specifier:

-c Create    -r Add/Replace  
-t List    -u Update    -x Extract

# BETTER

- › zip
- › unzip

FLAGS WITH  
FLAGS

# FLAGS HAVING SUB FLAGS

Extract: tar -x [options]

-k Keep existing files

-m Don't restore mod times

# FLAGS HAVING SUB FLAGS

Create: tar -c [options]

-z, -j, -J, --lzma

Compress archive with gzip/  
bzip2/xz/lzma

# INCOMPATIBLE FLAGS

# INCOMPATIBLE FLAGS

ls [options]

-S sort by file size

-t sort by modification time

-U do not sort

• • •

# INCOMPATIBLE FLAGS

ls -StU

*WHAT SHOULD THIS DO?*

# BETTER

```
> ls
```

```
--sort=[size,modtime,none]
```

# DOUBLE FLAGS

# FLAG --NO-FLAG

```
> git pull --stat --no-stat
```

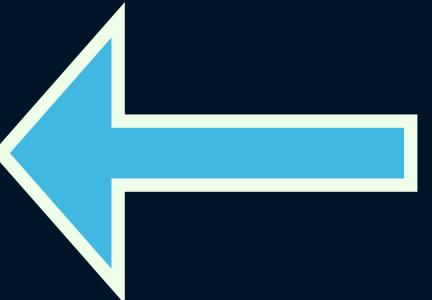
# BETTER

- › git pull --stat
- › git pull --stat=false

CLJ APPS

# CLI APPS

- › httpd
- › vi
- › emacs
- › git



*NOUN*

AN APP  
IS  
SOMETHING

# APPS

- Launch something
- Do more than one thing
- Collection of commands

# SUB COMMANDS

# SUB COMMANDS

- › svn add
- › brew install
- › npm search
- › apt-get upgrade
- › git clone

# SUB COMMANDS

- CLI apps do multiple things
- Apps are groups of commands
- (sub) Commands have flags & args
- All rules still apply

# EXPANDED RESOURCES

```
> brew    fetch    -v    hugo  
      ↑      ↑      ↑      ↑  
  APP    CMD    FLAG    ARG
```

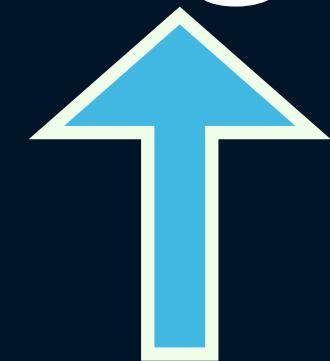
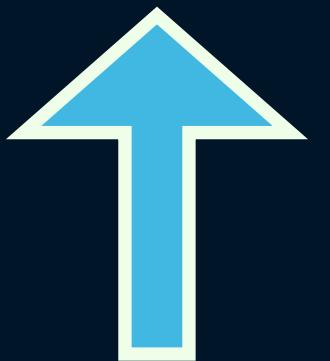
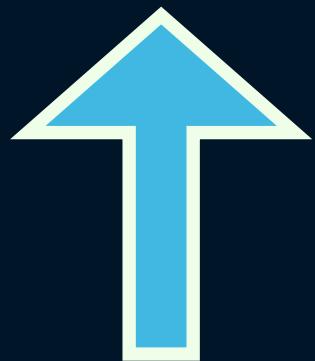
# NOUN

› brew install hugo

**NOUN**      **VERB**      **OBJECT**

# PRONOOUNCEABLE

› brew install hugo



*BREW, INSTALL HUGO*

# PRONOUNCEABLE

› brew

fetch

-v

hugo

NOUN

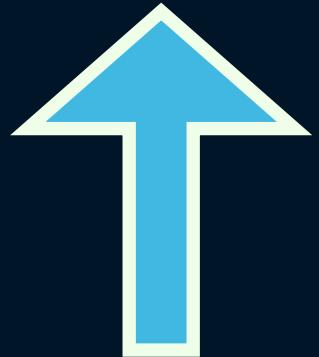
VERB

ADVERB

OBJECT

# PRONOUNCEABLE

› brew fetch -v hugo



*BREW, VERBOSELY*

*FETCH HUGO*

# 1. DESIGNING OUR APP

To Do...

WHAT'S IN  
A NAME?

**SHOCKING  
AMOUNT OF  
TODO APPS**

LET'S MAKE A  
BUNCH MORE

I'M CALLING  
MINE "TRI"

# FEATURES

# FEATURES

- Add Todo
- List Todos
- Mark "done"
- Search/Filter
- Priorities
- Archive
- Edit
- Create Dates
- Due Dates
- Tags
- Projects

# COMMAND LINE INTERFACE DESIGN

# CLI DESIGN IS EASY

- Requires:
  - no artistic talents
  - no special software
  - no special skills



~  
» |

OPEN A  
TERMINAL

# IMAGINE

- Think of how you would use your app
- Pretend your app was built
- Use it as if it existed
- Feel it out

# ADDING

# ADD

› tri add "Add Priorities"

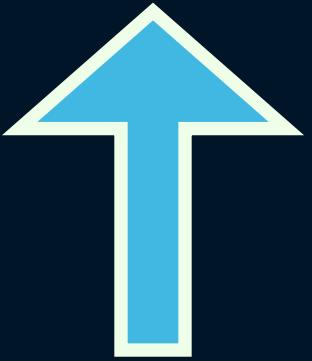
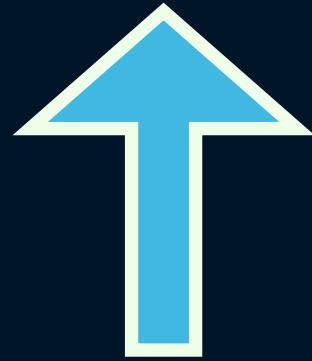
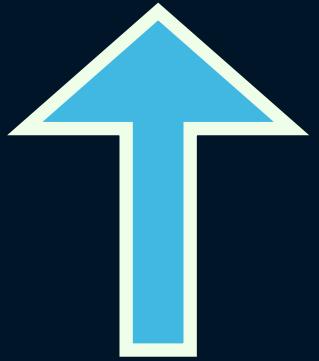
# ADD

› tri add Add Priorities

# ADD

› tri add "Add Priorities"

*NOUN*      *VERB*      *OBJECT*



# ADD

```
> tri add \
  "Add Multi Todo Support" \
  "Consider usage behaviors"
```

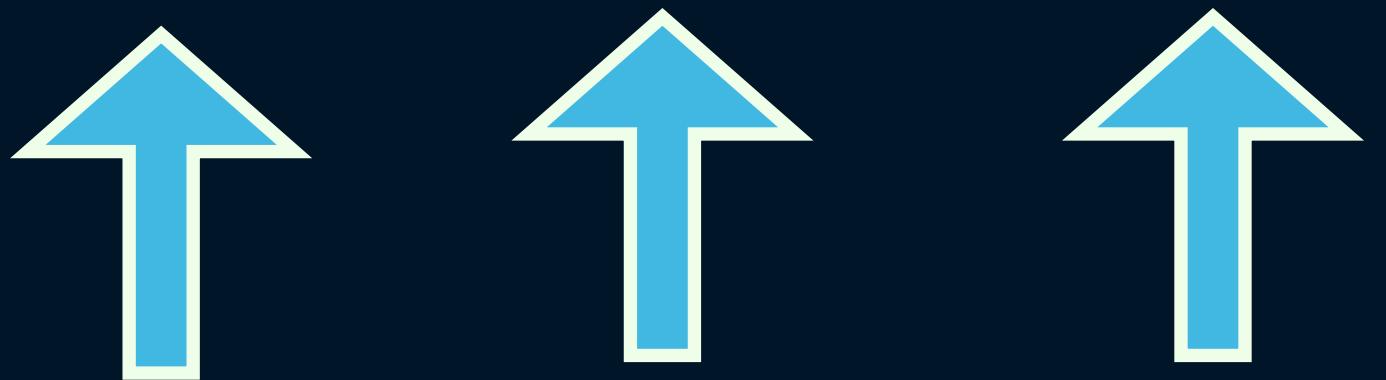
# PRIORITY

# ADD WITH PRIORITY

```
> tri add -p1 "Add listing"
```

# ADD WITH PRIORITY

```
> tri add -p1 "Add listing"
```



VERB  
NOUN ADVERB OBJECT

# PRONOUNCEABLE

```
> tri add -p1 "Add listing"
```



*TRI, ADD "ADD LISTING"  
TODO WITH A PRI OF 1*

# ALTERNATE SYNTAX

```
› tri add "Add listing P:1"
```

# CONSIDERATIONS

- What priority system to use?
  - Numeric
  - Alpha
  - High, Middle, Low
- What's the default priority?

# MY TODO

- High, Middle, Low
- H=1, L=3, M/\_=2
- Default is 2

# LISTING

# LISTING

› trials

# LISTING

› tri list

# LISTING

› tri

# LISTING OUTPUT

› tri list

(1) Add Listing

Consider usage behaviors

Add Multi Todo Support

Add Priorities

# LISTING OUTPUT

› tri list

(1) Add Listing

Add Priorities

Add Multi Todo Support

Consider usage behaviors

# LISTING OUTPUT

› tri list

(H) Add Listing

Consider usage behaviors

Add Priorities

(L) Add Multi Todo Support

# LISTING OUTPUT

› tri list

(H) Add Listing

Consider usage behaviors

Add Priorities

(L) Add Multi Todo Support

# LISTING OUTPUT

- › tri list
  - 1. (H) Add Listing
  - 2. Consider usage behaviors
  - 3. Add Priorities
  - 4. (L) Add Multi Todo Support

# FILTERING

# TOKENS

› tri list done

# FLAGS

```
> tri list -p1
```

# FILTER BY PROPERTY

- › tri list -p1
- › tri list --due June
- › tri list --created 12/15
- › tri list --done -p1

# FILTER BY PRIORITY

- › tri list -p2
- 2. Consider usage behaviors
- 3. Add Priorities

# SEARCHING

- › tri list "Add"
- 1. (H) Add Listing
- 3. Add Priorities
- 4. (L) Add Multi Todo Support

# SEARCHING

› tri list "Add" "Pri"

3. Add Priorities

# UPDATING

# LISTING OUTPUT

- › tri list
  - 1. (H) Add Listing
  - 2. Consider usage behaviors
  - 3. Add Priorities
  - 4. (L) Add Multi Todo Support

# COMPLETING

› tri done 2

# EDITING

› tri edit 1 "Improve Listing"

# EDITING

- › tri edit 1 -p2
- › tri edit 2 --due 05/13/15
- › tri edit 3 --created  
12/15

# CONSISTENCY

- › tri edit 1 -p2
- › tri list -p2
- › tri edit 3 --created 12/15
- › tri list --created 12/15

# BATCH EDIT?

```
> tri edit 1 2 3 -p2
```

GO

# INTRODUCTION

# WHY

# GO?

A LOT OF  
LANGUAGE EXIST,  
WHY DO WE NEED  
ANOTHER?

**WHAT LANGUAGE  
WOULD YOU USE  
TO WRITE A WEB  
SERVICE?**

# WEB SERVICE LANGS

- Python
- PHP
- Ruby
- Node.js

# WHY DYNAMIC?

- Developer Productivity
- Expressive
- High level
- Portable

**WHAT LANGUAGE  
WOULD YOU USE  
FOR A  
DATABASE?**

# DB LANGUAGES

- C
- C++
- Java

# STATIC

- High performance
- Precise
- Scalable
- Reliable

**WHAT IF YOU  
DIDN'T NEED  
TO CHOOSE?**

GO

# WHY GO

- Expressive
- Developer Productivity
- High level
- Portable

# WHY GO

- High Performance
- Precise
- Concurrent
- Good balance of control
- Super easy distribution & deployment

FRESH  
APPROACH

A photograph showing a close-up of a man's arm and shoulder, which has a tattoo of a compass rose. He is wearing a yellow shirt. In front of him, a tiger cub is lying on a wooden log, looking towards the camera. The background is blurred, suggesting an outdoor setting like a forest or jungle.

A NEW LANGUAGE  
IS LIKE A FOREIGN  
COUNTRY

A photograph of a group of people, mostly young women, gathered around a campfire at night. They are wearing traditional Korean Hanbok. Some have colorful face paint. They are smiling and holding sticks with marshmallows over the fire. The scene is warm and festive.

A NEW LANGUAGE  
IS LIKE A FOREIGN  
COUNTRY

A NEW LANGUAGE  
IS LIKE A FOREIGN  
COUNTRY

# FRESH APPROACH

- Go's design took a different approach
- Simplicity
- Reducing to minimum features



- Ken Thompson (B,C, Unix, UTF-8)
- Rob Pike (Unix, UTF-8)
- Robert Griesmier (Hotspot JVM, V8)

# FRESH APPROACH

- Not teaching about syntax
- Not explaining features
- Show code and talk about it
- Write code and talk about it

# 2. CREATING YOUR PROJECT

# COBRA



# cobra

- A CLI Command Framework
- A tool to generate CLI apps & commands
- Powers Kubernetes, Docker, Dropbox, Git Lfs, CoreOS, Hugo, Delve ...

# COBRA APP BUILDER

# COBRA INIT

```
> cobra init \  
github.com/<handle>/tri \  
-a "<Your Name>"
```

*REPLACE WITH YOUR URL,  
PROJECT NAME & NAME*

# GOPATH AND SETUP

- You WILL store your code in a GitHub user folder (example: `$GOPATH/src/github.com/username/helloworld`)
- I hated this :(
- I was comfortable with /code /dev/ or /projects

# THE SETUP

- The path really only matters if you are publishing this
- Start off assuming everything will be published

# GOPATH AND SETUP

- By Storing code on github, I was able to share my many problems
- Going from “I’m working on this” to “let’s collaborate”

# COBRA

```
› cobra init ...
```

Your Cobra application is  
ready at

/Users/spf13/gopath/src/  
github.com/<yourname>/tri

# CD TO PROJECT

```
> cd $GOPATH/src/  
github.com/<name>/tri
```

*REPLACE WITH YOUR URL,  
PROJECT NAME & NAME*

# LOOK AT YOUR PROJECT

› tree

```
•  
+-- LICENSE  
+-- cmd  
|   |-- root.go  
+-- main.go
```

# BUILD & RUN IT

- › go build
- › ./tri

A longer description that spans multiple lines  
and likely contains examples  
and usage of using your application. For  
example...

WHAT JUST  
HAPPENED?

YOU'VE JUST  
CREATED, BUILT &  
RAN YOUR 1ST  
GO APP

OPEN THE  
PROJECT IN  
AN EDITOR

# EDITORS

- No IDE needed - any text editor will do.
- Helpful features like autocomplete and format & import on-save
- If you're not sure what to use, I recommend VSCode — It's free, cross-platform, and easy to install.

New

Open...

Open Recent

Close Project

Project Structure... ⌘;

Other Settings ►

Import Settings...

Export Settings...

Settings Repository...

Save All ⌘S

Synchronize ⌘Y

Invalidate Caches / Restart...

Export to HTML...

Print...

Project...

Project from Existing Sources...

Project from Version Control ►

Module...

Module from Existing Sources...

Go File

File

Directory

HTML File

Resource Bundle



Project



tri ~ /gopath/src/github.com/spf13/tri

cmd

root.go

LICENSE

main.go

tri.iml

External Libraries

# EDITING OUR FIRST COMMAND

# CMD/ROOT.GO

```
package cmd

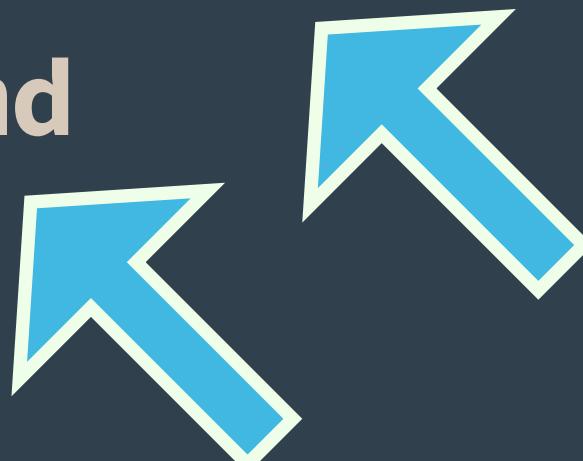
import (
    "fmt"
    "os"

    "github.com/spf13/cobra"
    "github.com/spf13/viper"
)
```

# PACKAGES

- Unit of organization of code in Go
- Organized by folders

# CMD/ROOT.GO

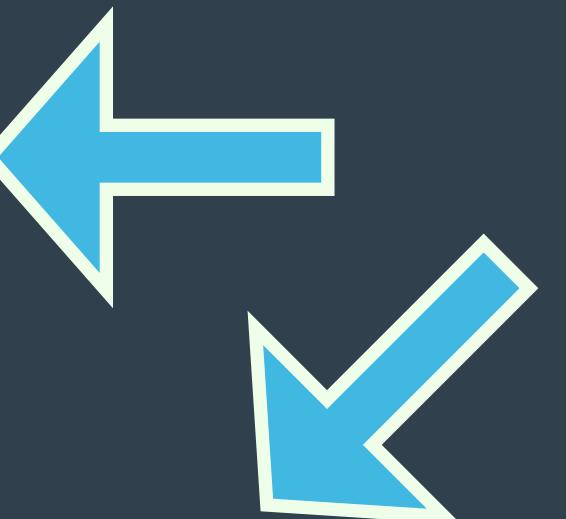
```
package cmd  
  
import (  
    "fmt"  
    "os"  
    "github.com/spf13cobra"  
    "github.com/spf13viper"  
)
```

*NOTICE PACKAGE  
NAME  
MATCHES DIR*

# CMD/ROOT.GO

```
package cmd

import (
    "fmt"
    "os"
    "github.com/spf13/cobra"
    "github.com/spf13/viper"
)
```



OTHER  
PACKAGES  
WE ARE  
IMPORTING

# CMD/ROOT.GO

```
// This represents the base command when called  
without any subcommands  
var RootCmd = &cobra.Command{  
    Use: "tri",  
    Short: "A brief description of your application",  
    Long: `A longer description that spans multiple  
    lines and likely ... application.`,  
}
```

# CMD/ROOT.GO

```
// This represents the base command when called  
without any subcommands  
var RootCmd = &cobra.Command{  
    Use:      "tri",  
    Short:    "Tri is a todo application",  
    Long:     `Tri will help you get more done in less time.  
It's designed to be as simple as possible to help you  
accomplish your goals.`,  
}
```

# CMD/ROOT.GO

// This represents the base command when called  
without any subcommands

```
var RootCmd = &cobra.Command{
```

Use: "tri",

Short: "Tri is a todo application",

Long: `Tri will help you get more done in less time.  
It's designed to be as simple as possible to help you  
accomplish your goals.`,

```
}
```

# CMD/ROOT.GO

```
var RootCmd = &cobra.Command{  
    Use: "tri",  
    Short: "Tri is a todo application",  
    Long: `Tri will help you get more done in  
    less time. VAR DECLARES  
    It's designed to be as simple as possible to  
    help you accomplish your goals.`,  
}
```



# CMD/ROOT.GO

```
var RootCmd = &cobra.Command{
```

Use: "tri",

Short: "Tri is a todo application",

Long: Tri will help you get more done in  
less time.

It's **designed** to be as simple as possible to  
help you accomplish your goals.`,

```
}
```

# CMD/ROOT.GO

```
var RootCmd = &cobra.Command{
```

Use: "tri",

Short: "Tri is a todo application",

Long: `Tri will help you get more done in  
less time.

It's designed to be as simple as possible to  
help you accomplish your goals.`,

```
}
```



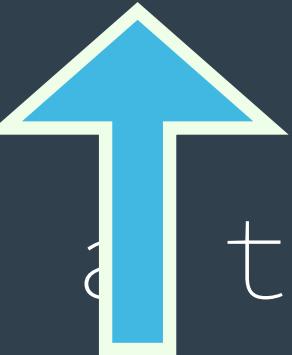
*INITIAL VALUE*

# CMD/ROOT.GO

```
var RootCmd = &cobra.Command{
```

Use:

"tri",



Short: "Tri is a todo application",

Long: `Tri will help you get more done in less time.

**INFERRED**

It's designed to be as simple as possible to help you accomplish your goals.

**FROM VALUE**

}

# ALTERNATE

```
// This represents the base command when called without any  
subcommands
```

```
var RootCmd *cobra.Command
```

```
RootCmd = &cobra.Command{  
    Use: "tri",  
    Short: "Tri is a todo application.",  
    Long: `Tri will help you get more done in less time.  
    It's designed to be as simple as possible to help you  
    accomplish your goals.`,  
}
```



**DECLARE w/o**

**INITIAL**

**VALUE**

# REDUNDANT

// This represents the base command when called without any subcommands

```
var RootCmd *cobra.Command = &cobra.Command{
```

Use: "tri",

Short: "Tri is a todo application",

Long: `Tri will help you get more done in less time.

It's designed to be as simple as possible to help you accomplish your goals. ,

```
}
```

**INITIAL VALUE**

# GO BUILD IT

- › go build
- › ./tri

Tri will help you get more done in less time.

It's designed to be as simple as possible to help you accomplish your goals.

HTTPS://  
GITHUB.COM  
/SPF13/TRI

# 3. CREATING OUR ADD COMMAND

ADD

"ADD!"

# CD TO PROJECT

```
> cd $GOPATH/src/  
github.com/spf13/tri
```

*REPLACE WITH YOUR URL,  
PROJECT NAME & NAME*

# COBRA ADD ADD

› cobra add add

add created at \$GOPATH/src/  
github.com/spf13/tri/cmd/  
add.go

# RUN "ADD"

```
> go build  
> ./tri add
```

add called

MAKE  
"ADD" ADD

# CMD/ADD.GO

package cmd

# CMD/ADD.GO

```
var addCmd = &cobra.Command{  
    Use: "add",  
    Short: "A brief description of your command",  
    Long: `A longer description that spans multiple  
    lines and likely...`,  
    Run: func(cmd *cobra.Command, args []string) {  
        // TODO: Work your own magic here  
        fmt.Println("add called")  
    },  
}
```

# CMD/ADD.GO

```
var addCmd = &cobra.Command{  
    Use: "add",  
    Short: "Add a new todo",  
    Long: `Add will create a new todo item to the list`,  
    Run: func(cmd *cobra.Command, args []string) {  
        // TODO: Work your own magic here  
        fmt.Println("add called")  
    },  
}
```

# FUNCTION

- Function is a type
- First class citizen in Go
- Can have multiple input values

# CMD/ADD.GO

```
var addCmd = &cobra.Command{
```

Use: "add",

Short: "Add a new todo",

Long: `Add will create a new todo  
item to the list`,

Run: **addRun**,

```
}
```

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command, args []string){  
    fmt.Println("add called")  
}  
}
```

# CMD/ADD.GO

```
var addRun = func(cmd *cobra.Command, args []string){  
    fmt.Println("add called")  
}
```

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command, args []string){
```



*WHAT IS THIS?*

```
}
```

# SLICE

- Ordered list of values of a single type
- Dynamic length
- 0 indexed

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command, args []string){
```

```
}
```



*WRITE SOMETHING  
TO PRINT EACH ARG*

# Effective Go

[Introduction](#)  
[Examples](#)  
[Formatting](#)  
[Commentary](#)  
[Names](#)

[Package names](#)  
[Getters](#)  
[Interface names](#)  
[MixedCaps](#)

[Semicolons](#)  
[Control structures](#)

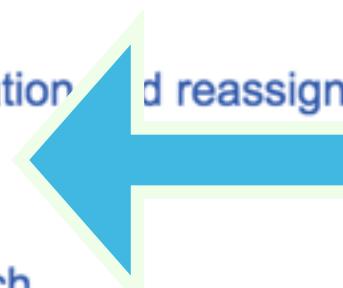
[If](#)  
[Redeclaration and reassignment](#)  
[For](#)  
[Switch](#)  
[Type switch](#)

[Functions](#)  
[Multiple return values](#)  
[Named result parameters](#)  
[Defer](#)

[Constants](#)  
[Variables](#)  
[The init function](#)  
[Methods](#)  
[Pointers vs. Values](#)  
[Interfaces and other types](#)  
[Interfaces](#)  
[Conversions](#)  
[Interface conversions and type assertions](#)  
[Generality](#)  
[Interfaces and methods](#)

[The blank identifier](#)  
[The blank identifier in multiple assignment](#)  
[Unused imports and variables](#)  
[Import for side effect](#)  
[Interface checks](#)

[Embedding](#)  
[Concurrency](#)  
[Share by communicating](#)  
[Goroutines](#)



# FOR

- Looper
- 3 parts: init; condition; post
- All parts are optional
- It is for, foreach & while from other languages

# FOR X, Y := RANGE

- Provides a way to iterate over an array, slice, string, map, or channel.
- x is index/key, y is value
- \_ allows you to ignore naming variables

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command, args []string){
```

```
}
```



*WRITE SOMETHING  
TO PRINT EACH ARG*

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command,  
args []string) {  
    for _, x := range args {  
        fmt.Println(x)  
    }  
}
```

# INIT()

- Special function
- Called after package variable declarations
- Each package may have multiple init()
- init() order un-guaranteed

# CMD/ADD.GO

```
func init() {  
    RootCmd.AddCommand(addCmd)  
}
```

# RUNNING ADD

```
> go build  
> ./tri add yoga "get milk"
```

yoga

get milk

MAIN

# MAIN.GO

```
package main

import "github.com/<yours>/tri/cmd"

func main() {
    cmd.Execute()
}

}
```

# MAIN MAIN MAIN

- Go executables are all about "main"
- Libraries don't have main
- 3 mains:
  - main.go (convention)
  - main package
  - main()
- main.main called after all init() are run

# MAIN.GO

```
package main

import "github.com/<yourself>/tri/cmd"

func main() { NAME
    cmd.Execute()
}

}
```

**PACKAGE**   
**NAME** 

# MAIN.GO

```
package main
```

```
import "github.com/<yours>/tri/cmd"
```

```
func main() {
```

```
    cmd.Execute()
```

```
}
```

**FUNCTION**

**NAME**



# CMD/ROOT.GO

```
func Execute() {  
    if err := RootCmd.Execute(); err != nil {  
        fmt.Println(err)  
        os.Exit(-1)  
    }  
}
```

# CMD/ROOT.GO

```
func Execute() {
    err := RootCmd.Execute()

    if err != nil {
        fmt.Println(err)
        os.Exit(-1)
    }
}
```

# CMD/ROOT.GO

```
func Execute() {  
    err := RootCmd.Execute()  
  
    if err != nil {  
        fmt.Println(err)  
        os.Exit(-1)  
    }  
}
```

# CMD/ROOT.GO

```
func Execute() {  
    err := RootCmd.Execute()  
  
    if err != nil {  
        fmt.Println(err)  
        os.Exit(-1)  
    }  
}  
}
```



**WHAT IS  
DOING?**



- Short Assignment operator
- Declares & assigns in one operation
- No type declaration

# CMD/ROOT.GO

```
func Execute() {  
    err := RootCmd.Execute()  
  
    if err != nil {  
        fmt.Println(err)  
        os.Exit(-1)  
    }  
}  
}
```

**WHAT IS ERR?**

# ERROR HANDLING

- Errors are not exceptional, they are **just values**
- No exceptions in Go
- Errors should be handled when they occur

# CMD/ROOT.GO

```
func Execute() {  
    err := RootCmd.Execute()  
  
    if err != nil {  
        fmt.Println(err)  
        os.Exit(-1)  
    }  
}
```

# CMD/ROOT.GO

```
func Execute() {  
    err := RootCmd.Execute()  
  
    if err != nil {  
        fmt.Println(err)  
        os.Exit(-1)  
    }  
}
```

# 4. CREATING OUR DATA MODEL

**CREATE A  
SECOND  
PACKAGE**

 Project ▾



▼  tri ~ /gopath/src/github.com/spf13/tri

▼  cmd

 add.go

 root.go

▼  todo

 todo.go

 LICENSE

 main.go

 tri.iml

**NEW FOLDER**

**NEW FILE**

# TODO/TODO.GO

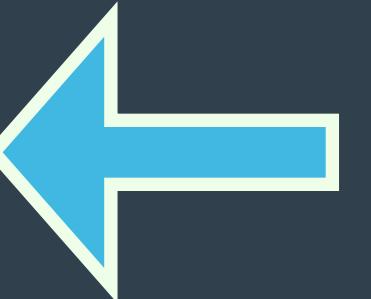
package todo

# TODO/TODO.GO

```
package todo
```

```
type Item struct {  
    Text string  
}
```

*WHAT ARE WE  
DOING HERE?*



# TODO/TODO.GO

```
package todo
```

```
type Item struct {  
    Text string  
}
```

# TODO/TODO.GO

```
package todo
```

```
type Item struct {  
    Text string  
}
```

# TODO/TODO.GO

```
package todo
```

```
type Item struct {  
    Text string  
}
```

# TODO/TODO.GO

```
package todo
```

```
type Item struct {
```

**Text string**

```
}
```

# NAMED TYPES

- Can be any known type (struct, string, int, slice, a new type you've declared, etc)
- Methods can be declared on it
- Not an alias - Explicit type

# TODO/TODO.GO

```
package todo
```

```
type Item struct {
```

```
    ↑ string
```

```
}
```

**EXPORTED**

# TODO/TODO.GO

```
package todo
```

```
type Item struct {
```

```
    Text string
```

```
}
```



**EXPORTED**

# ADDING ITEMS

# CMD/ADD.GO

```
import (  
    "fmt"  
  
    "github.com/spf13/cobra"  
  
    "github.com/<yourname>/tri/todo"  
)
```

*REPLACE WITH  
YOURS*



# CMD/ADD.GO

```
func addRun( ... ) {
```

```
for _, x := range args {
```

```
}
```

```
}
```

HOW DO WE  
CREATE A LIST  
(SLICE) OF TODOS?

# CONSTRUCTORS

- Go does not have constructors
- If initialization prior to use is needed use a factory
- Convention is to use `New_()`
- ... or `New()` when only one type is exported in the package

# COMPOSITE LITERALS

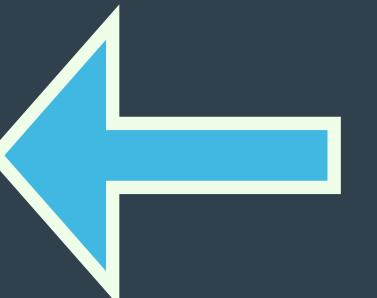
- An expression that creates a new value each time it is evaluated
- eg.. `[] todo.Item{}`

# CMD/ADD.GO

```
func addRun( . . . ) {  
    items := []todo.Item{}  
    for _, x := range args {  
        items = append(items, x)  
    }  
}
```

# CMD/ADD.GO

```
func addRun( . . . ) {  
    items := []todo.Item{}  
    for _, x := range args {  
        items = append(items, x) // ←  
    }  
    fmt.Println(items)  
}
```



*WHAT DO WE  
WANT  
TO DO?*

# APPEND

- Append adds new values to a slice
- Append will grow a slice as needed

# CMD/ADD.GO

```
func addRun( . . . ) {  
    items := []todo.Item{}  
    for _, x := range args {  
        items = append(items,  
    }  
    fmt.Println(items)  
}
```

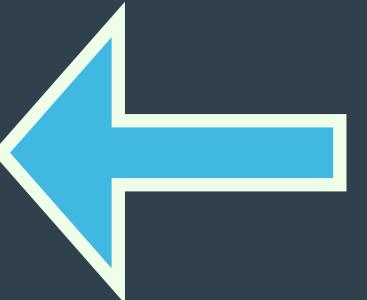
# CMD/ADD.GO

```
func addRun( . . . ) {  
    items := []todo.Item{}  
    for _, x := range args {  
        items = append(items,  
    }  
    fmt.Println(items)  
}
```

*WHAT DO  
WE APPEND?*

# CMD/ADD.GO

```
func addRun( . . . ) {  
    items := []todo.Item{}  
    for _, x := range args {  
        items = append(items,  
    }  
    fmt.Println(items)  
}
```



*REMEMBER  
THE TYPE*

# TODO/TODO.GO

```
package todo
```

```
type Item struct {  
    Text string  
}
```

# CMD/ADD.GO

```
func addRun( ... ) {  
    items := []todo.Item{}  
    for _, x := range args {  
        items = append(items, todo.Item{Text:x})  
    }  
    fmt.Println(items)  
}
```

# CMD/ADD.GO

```
func addRun(...) {
    items := []todo.Item{}
    for _, x := range args {
        items = append(items, todo.Item{Text:x})
    }
    fmt.Println(items)
}
```

# RUN "ADD"

```
> go build  
> ./tri add "one two" three  
[{one two} {three}]
```

# GORUN

```
> go run main.go add \
  "one two" three
[{one two} {three}]
```

# CMD/ADD.GO

```
func addRun( ... ) {  
    var items = []todo.Item{}  
    for _,x := range args {  
        items = append(items, todo.Item{Text:x})  
    }  
fmt.Printf("%#v\n", items)  
}
```

# GORUN

```
> go run main.go add \
"one two" three
```

```
[] todo.Item{todo.Item{Text:"one
two"}, todo.Item{Text:"three"}}}
```

# 5. PERSISTING OUR DATA

# HOW SHOULD WE PERSIST DATA?

# SAVING DATA TO A FILE

# TODO/TODO.GO

```
func SaveItems(filename string,  
    items []Item) error {  
  
    return nil  
}
```

# SERIALIZING DATA

[Go: encoding](#)

[Index](#) | [Files](#) | [Directories](#)

# package encoding

```
import "encoding"
```

Package encoding defines interfaces shared by other packages that convert data to and from byte-level and textual representations. Packages that check for these interfaces include encoding/gob, encoding/json, and encoding/xml. As a result, implementing an interface once can make a type useful in multiple encodings.

Standard types that implement these interfaces include time.Time and net.IP. The interfaces come in pairs that produce and consume encoded data.

## Index

[type BinaryMarshaler](#)

[type BinaryUnmarshaler](#)

[type TextMarshaler](#)

[type TextUnmarshaler](#)

## Package Files

[Go: encoding/json](#)[Index](#) | [Examples](#) | [Files](#)

# package json

```
import "encoding/json"
```

Package json implements encoding and decoding of JSON as defined in [RFC 4627](#). The mapping between JSON and Go values is described in the documentation for the Marshal and Unmarshal functions.

See "JSON and Go" for an introduction to this package: [https://golang.org/doc/articles/json\\_and\\_go.html](https://golang.org/doc/articles/json_and_go.html)

## Index

- func `Compact(dst *bytes.Buffer, src []byte) error`
- func `HTMLEscape(dst *bytes.Buffer, src []byte)`
- func `Indent(dst *bytes.Buffer, src []byte, prefix, indent string) error`
- func `Marshal(v interface{}) ([]byte, error)`
- func `MarshalIndent(v interface{}, prefix, indent string) ([]byte, error)`
- func `Unmarshal(data []byte, v interface{}) error`
- type `Decoder`
  - func `NewDecoder(r io.Reader) *Decoder`
  - func (`func (d *Decoder) ReadJSON(f io.Reader)`)

**EXERCISE:  
WRITE A  
SERIALIZATION  
FUNCTION TO JSON**

# TODO/TODO.GO

```
import (  
    "encoding/json"  
)
```

# TODO/TODO.GO

```
func SaveItems(filename string, items []Item) error {
```

```
}
```

# TODO/TODO.GO

```
func SaveItems(filename string, items []Item) error {
    b, err := json.Marshal(items)
    if err != nil {
        return err
    }

    fmt.Printf("%s\n", b)

    return nil
}
```

# TODO/TODO.GO

```
func SaveItems(filename string, items []Item) error {
    b, err := json.Marshal(items)
    if err != nil {
        return err
    }

    fmt.Printf("%s\n", b)

    return nil
}
```

# TODO/TODO.GO

```
func SaveItems(filename string, items []Item) error {
    b, err := json.Marshal(items)
    if err != nil {
        return err
    }

    fmt.Printf("%s\n", b)

    return nil
}
```

# TODO/TODO.GO

```
func SaveItems(filename string, items []Item) error {
    b, err := json.Marshal(items)
    if err != nil {
        return err
    }

    fmt.Printf("%s\n", b)

    return nil
}
```

HAVE ADD CMD  
CALL THIS NEW  
FUNCTION

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command,
            args []string) {
    var items = []todo.Item{}
    for _, x := range args {
        items = append(items, todo.Item{Text: x})
    }
    todo.SaveItems("x", items)
}
```

# CHECK JSON CREATION

```
>go run main.go add \
  "one two" three
[{"Text": "one two"},  
 {"Text": "three"}]
```

# WRITING TO FILES

HOW DO WE  
FIGURE OUT HOW  
TO WRITE TO  
FILES IN GO?



write to file in golang



Sign in

SafeSearch on



golang **append** to file

golang **overwrite** file

golang **delete** file

golang **check if file exists**

About 421,000 results (0.49 seconds)

## Go by Example: Writing Files

<https://gobyexample.com/writing-files> ▾

Writing files in Go follows similar patterns to the ones we saw earlier for reading.  
package main ... To start, here's how to dump a string (or just bytes) into a file.

## go - How to read/write from/to file? - Stack Overflow

[stack overflow .com/questions/1821811/how-to-read-write-from-to-file](https://stackoverflow.com/questions/1821811/how-to-read-write-from-to-file) ▾

30 Nov 2009 - Let's make a Go 1-compatible list of all the ways to read and write files in  
Go. ... In the following examples I copy a file by reading from it and writing to the  
destination file. Start with the basics package main import ( "io" "os" ) ...

## ioutil - The Go Programming Language

<https://golang.org/pkg/io/ioutil/> ▾

File, err error): func WriteFile(filename string, data []byte, perm os.FileMode) error ... is  
an io.Writer on which all Write calls succeed without doing anything.

## Writing to File – Golang Code



# awesome

Awesome Go

build passing

😎 awesome

chat on gitter

Go

Documents

Packages

The Project

Help

Blog

Play

Search

# Packages

[Standard library](#)

[Other packages](#)

[Sub-repositories](#)

[Community](#)

## Standard library

Name	Synopsis
<a href="#">archive</a>	
<a href="#">tar</a>	Package tar implements access to tar archives.
<a href="#">zip</a>	Package zip provides support for reading and writing ZIP archives.
<a href="#">bufio</a>	Package bufio implements buffered I/O. It wraps an io.Reader or io.Writer object, creating another object (Reader or Writer) that also implements the interface but provides buffering and some help for textual I/O.
<a href="#">builtin</a>	Package builtin provides documentation for Go's predeclared identifiers.



# STANDARD LIBRARY

Let's be honest, you don't want to write everything from scratch so most programming depends on your ability to interface with existing libraries. i.e. packages. Here are a few core packages you should know about;

- Strings
- Input/Output (io/ioutil)
- Errors
- fmt
- Containers and Sort
- path/filepath
- HTTP (net/http)
- math/rand

[Go](#)[Documents](#)[Packages](#)[The Project](#)[Help](#)[Blog](#)[Play](#)[Search](#)

# Package ioutil

```
import "io/ioutil"
```

[Overview](#)[Index](#)[Examples](#)

## Overview ▾

Package ioutil implements some I/O utility functions.

## Index ▾

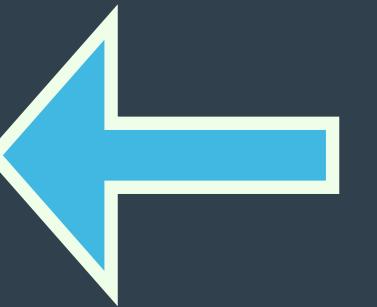
[Variables](#)[func NopCloser\(r io.Reader\) io.ReadCloser](#)

# TODO/TODO.GO

```
import (
    "io/ioutil"
    "encoding/json"
)
```

# TODO/TODO.GO

```
func SaveItems(filename string, items  
[]Item) error {  
    ...  
    return nil  
}
```



*EXERCISE  
USE IOUTIL &  
WRITE A FILE*

# TODO/TODO.GO

```
func SaveItems(filename string, items []Item) error {  
    ...  
    err = ioutil.WriteFile(filename, b, 0644)  
    if err != nil {  
        return err  
    }  
  
    return nil  
}
```

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command,
            args []string) {
    var items = []todo.Item{}
    for _, x := range args {
        items = append(items, todo.Item{Text: x})
    }

    err := todo.SaveItems("/Users/spf13/.tridos.json", items);
    if err != nil {
        fmt.Errorf("%v", err)
    }
}
```

# CHECK FILE CREATION

```
› go run main.go add "one two" three  
› cat ~/.tridos.json  
[{"Text": "one two"},  
 {"Text": "three"}]
```

# READING DATA

NEXT WRITE  
READITEMS  
FUNCTION IN  
TODO.GO

# TODO/TODO.GO

```
func ReadItems(filename string) ([]Item, error) {  
    return []Item{}, nil  
}
```

# TODO/TODO.GO

```
func ReadItems(filename string) ([]Item, error) {  
    return []Item{}, nil  
}
```



*TWO RETURN  
VALUES*

**EXERCISE:**  
**USE IOUTIL & JSON TO**  
**POPULATE READITEMS**  
**FUNCTION**

# POINTERS / REFERENCES

- Go functions operate on copies of values
- Go pointers enable "pass by reference" behavior
- Functions operating on pointers mutate the value

# TODO/TODO.GO

```
func ReadItems(filename string) ([]Item, error) {  
    b, err := ioutil.ReadFile(filename)  
    if err != nil {  
        return []Item{}, err  
    }  
    ...  
}
```

# TODO/TODO.GO

```
func ReadItems(filename string) ([]Item, error) {  
    b, err := ioutil.ReadFile(filename)  
    if err != nil {  
        return []Item{}, err  
    }  
    ...  
}
```

# TODO/TODO.GO

```
func ReadItems(filename string) ([]Item, error) {  
    ...  
var items []Item  
    if err := json.Unmarshal(b, &items); err != nil {  
        return []Item{}, err  
    }  
    return items, nil  
}
```

# TODO/TODO.GO

```
func ReadItems(filename string) ([]Item, error) {  
    ...  
    var items []Item  
    if err := json.Unmarshal(b, &items); err != nil {  
        return []Item{}, err  
    }  
    return items, nil  
}
```

# TODO/TODO.GO

```
func ReadItems(filename string) ([]Item, error) {  
    ...  
    var items []Item  
    if err := json.Unmarshal(b, &items); err != nil {  
        return []Item{}, err  
    }  
    return items, nil  
}
```

# TODO/TODO.GO

```
func ReadItems(filename string) ([]Item, error) {  
    ...  
    var items []Item  
    if err := json.Unmarshal(b, &items); err != nil {  
        return []Item{}, err  
    }  
    return items, nil  
}
```

# TODO/TODO.GO

```
func ReadItems(filename string) ([]Item, error) {  
    ...  
    var items []Item  
    if err := json.Unmarshal(b, &items); err != nil {  
        return []Item{}, err  
    }  
return items, nil  
}
```

# TODO/TODO.GO

```
func ReadItems(filename string) ([]Item, error) {  
    ...  
    var items []Item  
    err := json.Unmarshal(b, &items);  
  
    if err != nil {  
        return []Item{}, err  
    }  
  
return items, nil  
}
```

MAKE  
ADDREAD

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command, args []string) {  
    var items = []todo.Item{}  
  
    for _, x := range args {  
        items = append(items, todo.Item{Text: x})  
    }  
  
    ...  
}
```

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command, args []string) {
    items, err := todo.ReadItems("/Users/spf13/.tridos.json")
    if err != nil {
        log.Printf("%v", err)
    }

    for _, x := range args {
        items = append(items, todo.Item{Text: x})
    }
}

...
```

# GO FMT

- Formats your go code for you
- Awesome to do "on save"
- End of all stylistic debates

# GO FMT

```
> go fmt ./cmd
```

# LIST COMMAND

**EXERCISE:**  
**COMBINE ALL WE'VE**  
**LEARNED TO ADD**  
**THE LIST COMMAND**

## README.md



# cobra

 Share Image

Cobra is both a library for creating powerful modern CLI applications as well as a program to generate applications and command files.

Many of the most widely used Go projects are built using Cobra including:

- [Kubernetes](#)
- [Hugo](#)

```
tri git/master* 11s
> godoc github.com/spf13/cobra
use 'godoc cmd/github.com/spf13/cobra' for documentation on the github.com/spf13/cobra command
```

## PACKAGE DOCUMENTATION

```
package cobra
import "github.com/spf13/cobra"
```

Package cobra is a commander providing a simple interface to create powerful modern CLI interfaces. In addition to providing an interface, Cobra simultaneously provides a controller to organize your application code.

## CONSTANTS

```
const (
    BashCompFilenameExt      = "cobra_annotation_bash_completion_filename_extensions"
    BashCompCustom           = "cobra_annotation_bash_completion_custom"
    BashCompOneRequiredFlag = "cobra_annotation_bash_completion_one_required_flag"
    BashCompSubdirsInDir     = "cobra_annotation_bash_completion_subdirs_in_dir"
)
```

## VARIABLES

```
var EnableCommandSorting = true
    EnableCommandSorting controls sorting of the slice of commands which is
```

# ADD LIST COMMAND

```
› cobra add list
```

```
list created at $GOPATH/  
src/github.com/spf13/tri/  
cmd/list.go
```

# CMD/LIST.GO

```
Run: func(cmd *cobra.Command, args []string) {
    items, err := todo.ReadItems("/Users/
spf13/.tridos.json")

    if err != nil {
        log.Printf("%v", err)
    }
    fmt.Println(items)
},
```

# CMD/LIST.GO

```
Run: func(cmd *cobra.Command, args []string) {
    items, err := todo.ReadItems("/Users/
spf13/.tridos.json")

    if err != nil {
        log.Printf("%v", err)
    }
    fmt.Println(items)
},
```

# CMD/LIST.GO

```
Run: func(cmd *cobra.Command, args []string) {
    items, err := todo.ReadItems("/Users/
spf13/.tridos.json")

    if err != nil {
        log.Printf("%v", err)
    }
    fmt.Println(items)
},
```

# RUN LIST

```
> go run main.go list
```

```
[{one two} {three}]
```

# 6. ADDING ROOT (GLOBAL) FLAGS

USING  
\$HOME

# GO GET

```
> go get github.com/  
mitchellh/go-homedir
```

# CMD/ROOT.GO

```
package cmd
```

```
import (
```

```
    ...
```

```
"github.com/mitchellh/go-homedir"
```

```
)
```

# CMD/ROOT.GO

```
var dataFile string
```

# CMD/ROOT.GO

```
func init() {  
    // Here you will define your flags and configuration settings  
    // Cobra supports Persistent Flags which if defined here will  
be global for your application  
  
    home, err := homedir.Dir()  
    if err != nil {  
        log.Println("Unable to detect home directory. Please set  
data file using --datafile.")  
    }  
    ...  
}
```

ADDING

A PLAG

# CMD/ROOT.GO

```
func init() {  
    ...  
    RootCmd.PersistentFlags().StringVar(&dataFile,  
        "datafile",  
        home+string(os.PathSeparator)+".tridos.json",  
        "data file to store todos")  
    ...  
}
```

# CMD/ROOT.GO

```
func init() {  
    ...  
  
    RootCmd.PersistentFlags().StringVar(&dataFile,  
        "datafile",  
        home+string(os.PathSeparator)+".tridos.json",  
        "data file to store todos")  
}
```

# CMD/ROOT.GO

```
func init() {  
    ...  
  
    RootCmd.PersistentFlags().StringVar(&dataFile,  
        "datafile",  
        home+string(os.PathSeparator)+"tridos.json",  
        "data file to store todos")  
}
```

# CMD/ROOT.GO

```
func init() {  
    ...  
  
    RootCmd.PersistentFlags().StringVar(&dataFile,  
        "datafile",  
        home+string(os.PathSeparator)+".tridos.json",  
        "data file to store todos")  
}
```

# CMD/ADD.GO

```
items, err := todo.ReadItems(dataFile)
...
err := todo.SaveItems(dataFile, items)
```

# CMD/LIST.GO

```
items, err := todo.ReadItems(dataFile)
```

# SEE THE FLAG

```
> go build  
> ./tri
```

...

Flags:

```
  --datafile string  data file to store todos (default "/  
Users/spf13/.tridos.json")  
-h, --help          help for tri
```

Use "tri [command] --help" for more information about a command.

# USE THE FLAG

- › go build
- › ./tri add "Add priorities" \  
--datafile \$HOME/.next.json
- › ./tri list --datafile \$HOME/.next.json  
[{Add priorities}]

# 7. ADDING PRIORITIES

# ADD SOME TODOS

```
› go run main.go add \
  "add priorities" \
  "order by priority"
```

# ADJUSTING OUR TYPE

# TODO/TODO.GO

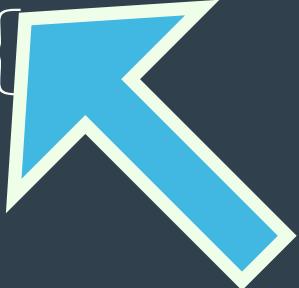
```
type Item struct {
    Text      string
    Priority int
}
```

# VALIDATING INPUT

# TODO/TODO.GO

```
func (i *Item) SetPriority(pri int) {  
    switch pri {  
        case 1:  
            i.Priority = 1  
        case 3:  
            i.Priority = 3  
        default:  
            i.Priority = 2  
    }  
}
```

# TODO/TODO.GO

```
func (i *Item) SetPriority(pri int) {  
    switch pri {  
        case 1:  METHOD  
            i.Priority = 1  
        case 3:  
            i.Priority = 3  
        default:  
            i.Priority = 2  
    }  
}
```

# TODO/TODO.GO

```
func (i *Item) SetPriority(pri int) {  
    switch pri {  
        case 1:  
            i.Priority = 1  
        case 3:  
            i.Priority = 3  
        default:  
            i.Priority = 2  
    }  
}
```

# TODO/TODO.GO

```
func (i *Item) SetPriority (pri int) {  
    switch pri {  
        case 1:  
            i.Priority = 1  
        case 3:  
            i.Priority = 3  
        default:  
            i.Priority = 2  
    }  
}
```

# ALTERNATIVE

```
func (i *Item) SetPriority (pri int) {  
    switch pri {  
        case 1,3:  
            i.Priority = pri  
        default:  
            i.Priority = 2  
    }  
}
```

# ADDING

# FLAG

# CMD/ADD.GO

```
var priority int  
.  
func init() {  
    RootCmd.AddCommand(addCmd)  
  
    addCmd.Flags().IntVarP(&priority,  
        "priority", "p", 2, "Priority:1,2,3")  
.  
.
```

# CMD/ADD.GO

```
var priority int  
.  
.  
func init() {  
    RootCmd.AddCommand(addCmd)  
  
    addCmd.Flags().IntVarP(&priority,  
        "priority", "p", 2, "Priority:1,2,3")  
.  
.
```

# HELP ADD

› go build

› ./tri help add

Add will create a new todo item to the list

Usage:

tri add [flags]

Flags:

-p, --priority int    Priority:1,2,3 (default 2)

# SETTING

# PRIORITY

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command, args []string) {  
    ...  
    for _, x := range args {  
        item := todo.Item{Text: x}  
        item.SetPriority(priority)  
        items = append(items, item)  
    }  
}
```

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command, args []string) {  
    ...  
    for _, x := range args {  
        item := todo.Item{Text: x}  
        item.SetPriority(priority)  
        items = append(items, item)  
    }  
}
```

 CALLING OUR  
METHOD

# ADD WITH PRIORITY

```
> ./tri add "format list" -p1
```

```
> ./tri list
```

```
[{add priorities 0} {order by priority 0} {format list 1}]
```

# 8. MAKING OUR LIST PRETTY

# BREAK OUT LISTRUN

# CMD/LIST.GO

```
// listCmd respresents the list command
var listCmd = &cobra.Command{
    Use:   "list",
    Short: "List the todos",
    Long:  `Listing the todos`,
    Run:   listRun,
}

func listRun(cmd *cobra.Command, args []string) {
    ...
}
```

TAB  
WRITER

# CMD/LIST.GO

```
func listRun(cmd *cobra.Command, args []string) {  
    ...  
    w := tabwriter.NewWriter(os.Stdout, 3, 0, 1, ' ', 0)  
    for _, i := range items {  
        fmt.Fprintln(w, strconv.Itoa(i.Priority)+"\t"+i.Text+"\t")  
    }  
  
    w.Flush()  
}
```

# CMD/LIST.GO

```
func listRun(cmd *cobra.Command, args []string) {  
    ...  
    w := tabwriter.NewWriter(os.Stdout, 3, 0, 1, ' ', 0)  
    for _, i := range items {  
        fmt.Fprintln(w, strconv.Itoa(i.Priority)+"\t"+i.Text+"\t")  
    }  
  
    w.Flush()  
}
```

# CMD/LIST.GO

```
func listRun(cmd *cobra.Command, args []string) {  
    ...  
    w := tabwriter.NewWriter(os.Stdout, 3, 0, 1, ' ', 0)  
    for _, i := range items {  
        fmt.Fprintln(w, strconv.Itoa(i.Priority)+"\t"+i.Text+"\t")  
    }  
  
    w.Flush()  
}
```

# CMD/LIST.GO

```
func listRun(cmd *cobra.Command, args []string) {  
    ...  
    w := tabwriter.NewWriter(os.Stdout, 3, 0, 1, ' ', 0)  
    for _, i := range items {  
        fmt.Fprintln(w, strconv.Itoa(i.Priority)+"\t"+i.Text+"\t")  
    }  
  
    w.Flush()  
}
```

# CMD/LIST.GO

```
func listRun(cmd *cobra.Command, args []string) {  
    ...  
    w := tabwriter.NewWriter(os.Stdout, 3, 0, 1, ' ', 0)  
    for _, i := range items {  
        fmt.Fprintln(w, strconv.Itoa(i.Priority)+"\t"+i.Text+"\t")  
    }  
  
    w.Flush()  
}
```

# LIST W/PRIORITY

- › go run main.go list
- 0 add priorities
- 0 order by priority
- 1 format list

PRETTY  
PRIORITY  
PRINTING

# TODO/TODO.GO

```
func (i *Item) PrettyP() string {
    if i.Priority == 1 {
        return "(1)"
    }
    if i.Priority == 3 {
        return "(3)"
    }
    return " "
}
```

# CMD/LIST.GO

```
func listRun(cmd *cobra.Command, args []string) {  
    ...  
    w := tabwriter.NewWriter(os.Stdout, 3, 0, 1, ' ', 0)  
    for _, i := range items {  
        fmt.Fprintln(w, i.PrettyP()+"\t"+i.Text+"\t")  
    }  
  
    w.Flush()  
}
```

# LIST W/PRIORITY

› go run main.go list

add priorities

order by priority

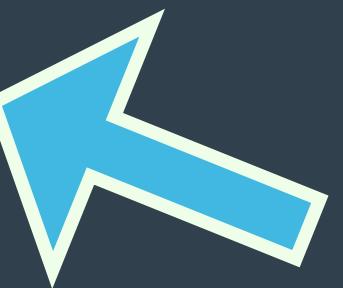
(1) format list

LABELS

# TODO/TODO.GO

```
type Item struct {
    Text      string
    Priority int
position int
}
```

# TODO/TODO.GO

```
type Item struct {  
    Text      string  
    Priority int  
    position int  
}  

```

*NOTICE  
LOWERCASE P*

# TODO/TODO.GO

```
func (i *Item) Label() string {  
    return strconv.Itoa(i.position) + "."  
}
```

# TODO/TODO.GO

```
func ReadItems(filename string) ([]Item, error) {  
    ...  
  
    for i, _ := range items {  
        items[i].position = i + 1  
    }  
  
    return items, nil  
}
```

# LIST W/PRIORITY

- › go run main.go list
- 1. add priorities
- 2. order by priority
- 3. (1) format list

SORT

# TODO/TODO.GO

```
// ByPri implements sort.Interface for []Item based on  
// the Priority & position field.
```

```
type ByPri []Item
```

```
func (s ByPri) Len() int { return len(s) }  
func (s ByPri) Swap(i, j int) { s[i], s[j] = s[j], s[i] }  
func (s ByPri) Less(i, j int) bool {  
    if s[i].Priority == s[j].Priority {  
        return s[i].position < s[j].position  
    }  
    return s[i].Priority < s[j].Priority  
}
```

# TODO/TODO.GO

```
// ByPri implements sort.Interface for []Item based on
// the Priority & position field.

type ByPri []Item

func (s ByPri) Len() int          { return len(s) }
func (s ByPri) Swap(i, j int)     { s[i], s[j] = s[j], s[i] }
func (s ByPri) Less(i, j int) bool {
    if s[i].Priority == s[j].Priority {
        return s[i].position < s[j].position
    }
    return s[i].Priority < s[j].Priority
}
```

# TODO/TODO.GO

```
// ByPri implements sort.Interface for []Item based on
// the Priority & position field.
type ByPri []Item

func (s ByPri) Len() int          { return len(s) }
func (s ByPri) Swap(i, j int)     { s[i], s[j] = s[j], s[i] }
func (s ByPri) Less(i, j int) bool {
    if s[i].Priority == s[j].Priority {
        return s[i].position < s[j].position
    }
    return s[i].Priority < s[j].Priority
}
```

# TODO/TODO.GO

```
// ByPri implements sort.Interface for []Item based on
// the Priority & position field.

type ByPri []Item

func (s ByPri) Len() int          { return len(s) }
func (s ByPri) Swap(i, j int) { s[i], s[j] = s[j], s[i] }
func (s ByPri) Less(i, j int) bool {
    if s[i].Priority == s[j].Priority {
        return s[i].position < s[j].position
    }
    return s[i].Priority < s[j].Priority
}
```

# TODO/TODO.GO

```
// ByPri implements sort.Interface for []Item based on
// the Priority & position field.
type ByPri []Item

func (s ByPri) Len() int          { return len(s) }
func (s ByPri) Swap(i, j int) { s[i], s[j] = s[j], s[i] }
func (s ByPri) Less(i, j int) bool {
    if s[i].Priority == s[j].Priority {
        return s[i].position < s[j].position
    }
    return s[i].Priority < s[j].Priority
}
```

# TODO/TODO.GO

```
// ByPri implements sort.Interface for []Item based on
// the Priority & position field.

type ByPri []Item

func (s ByPri) Len() int          { return len(s) }
func (s ByPri) Swap(i, j int)     { s[i], s[j] = s[j], s[i] }
func (s ByPri) Less(i, j int) bool {
    if s[i].Priority == s[j].Priority {
        return s[i].position < s[j].position
    }
return s[i].Priority < s[j].Priority
}
```

# CMD/LIST.GO

```
func listRun(cmd *cobra.Command, args []string) {  
    ...  
  
sort.Sort(todo.ByPri(items))  
  
    w := tabwriter.NewWriter(os.Stdout, 3, 0, 1, ' ', 0)  
    ...
```

# LIST W/PRIORITY

- › go run main.go list
- 3. (1) format list
- 1. add priorities
- 2. order by priority

# 9. DONE-ING TODOS

ADD  
DONE

# COBRA ADD DONE

› cobra add done

done created at \$GOPATH/  
src/github.com/spf13/tri/  
cmd/done.go

# CMD/DONE.GO

```
// doneCmd represents the done command
var doneCmd = &cobra.Command{
    Use:      "done",
    Aliases: []string{"do"},
    Short:    "Mark Item as Done",
    Run:     doneRun,
}
```

ADD  
DONE

# TODO/TODO.GO

```
type Item struct {
    Text      string
    Priority int
    position int
}
```

# TODO/TODO.GO

```
type Item struct {  
    Text      string  
    Priority int  
    position int  
    Done      bool  
}
```

TRI DONE  
SET  
.DONE

# CMD/DONE.GO

```
func doneRun(cmd *cobra.Command, args []string) {
    items, err := todo.ReadItems(dataFile)
    i, err := strconv.Atoi(args[0])

    if err != nil {
        log.Fatalln(args[0], "is not a valid label\n", err)
    }
}
```

...

Break out still

# CMD/DONE.GO

```
func doneRun(cmd *cobra.Command, args []string) {  
    ...  
    if i > 0 && i < len(items) {  
        items[i-1].Done = true  
        fmt.Printf("%q %v\n", items[i-1].Text, "marked done")  
  
        sort.Sort(todo.ByPri(items))  
        todo.SaveItems(dataFile, items)  
    } else {  
        log.Println(i, "doesn't match any items")  
    }  
}
```

Break out slide

# DONE WRONG

› go run main.go done a

2016/05/14 22:22:03 a is not  
a valid label

strconv.ParseInt: parsing

"a": invalid syntax

# DONE WRONG

› go run main.go done 13

2016/05/14 22:44:06 13

doesn't match any items

# DONE RIGHT

```
> go run main.go done 1
```

# UPDATE LISTING

402

spf13

# TODO/TODO.GO

```
func (i *Item) PrettyDone() string {  
    if i.Done {  
        return "X"  
    }  
    return ""  
}
```

# CMD/LIST.GO

```
fmt.Fprintln(w, i.Label()  
+"\\t"+i.PrettyDone()  
+"\\t"+i.PrettyP()  
+"\\t"+i.Text+"\\t")
```

# TODO/TODO.GO

```
func (s ByPri) Less(i, j int) bool {
    if s[i].Done != s[j].Done {
        return s[i].Done
    }

    if s[i].Priority != s[j].Priority {
        return s[i].Priority < s[j].Priority
    }

    return s[i].Position < s[j].Position
}
```

Rewritten

# LIST

› go run main.go list

1. (1) format list
2. X order by priority
3. X add priorities

ADD  
--DONE  
FLAG

# ADD NEW TODO

```
> go run main.go add "hide  
done items"
```

# CMD/LIST.GO

```
fmt.Fprintln(w, i.Label()  
+"\\t"+i.PrettyDone()  
+"\\t"+i.PrettyP()  
+"\\t"+i.Text+"\\t")
```

# CMD/LIST.GO

```
var (
    doneOpt bool
)
```

# CMD/LIST.GO

```
func init() {
    RootCmd.AddCommand(listCmd)

    listCmd.Flags().BoolVar(&doneOpt,
        "done", false, "Show 'Done' Todos")
}
```

# CMD/LIST.GO

```
func listRun(cmd *cobra.Command, args []string) {  
    ...  
  
    for _, i := range items {  
        if i.Done == doneOpt {  
            fmt.Fprintln(w, i.Label()...)  
        }  
    }  
}
```

# TRY --DONE

- go run main.go list --done
- 3. X order by priority
- 4. X add priorities

SHOW

AII

# CMD/LIST.GO

```
var (
    doneOpt bool
    allOpt  bool
)
```

# CMD/LIST.GO

```
func init() {  
    ...  
    listCmd.Flags().BoolVar(&allOpt,  
        "all", false, "Show all Todos")  
}
```

# CMD/LIST.GO

```
func listRun(cmd *cobra.Command, args []string) {  
    ...  
  
    for _, i := range items {  
        if allopt || i.Done == doneOpt {  
            fmt.Fprintln(w, i.Label()...)  
        }  
    }  
}
```

# TRY --ALL

- › go run main.go list --all --done
- 1. (1) format list
- 2. hide done items
- 3. X order by priority
- 4. X add priorities

# 10. CONFIG

# USING DIFFERENT DATA FILE

```
>./tri list --datafile \
```

```
$HOME/Dropbox/Sync/tridos.json
```

```
>./tri add "Add config/ENV support" \
```

```
--datafile $HOME/Dropbox/Sync/tridos.json
```

# CONFIG

## FTW

# viper



- A configuration manager
- Registry for application settings
- Works well with Cobra

# VIPER SUPPORTS

- YAML, TOML, JSON or HCL
- Etcd, Consul
- Config LiveReloading
- default < KeyVal < config < env < flag
- Aliases & Nested values

# CMD/ROOT.GO

```
var cfgFile string
```

**ALREADY THERE**

```
func init() {
```

```
    cobra.OnInitialize(initConfig)
```

```
...
```

```
    RootCmd.PersistentFlags().StringVar(&cfgFile,  
    "config", "", "config file (default is  
    $HOME/.tri.yaml)")
```

```
...
```

# CMD/ROOT.GO

```
// Read in config file and ENV variables if set.  
func initConfig() {  
    viper.SetConfigName(".tri")  
    viper.AddConfigPath("$HOME")  
    viper.AutomaticEnv()  
  
    // If a config file is found, read it in.  
    if err := viper.ReadInConfig(); err == nil {  
        fmt.Println("Using config file:", viper.ConfigFileUsed())  
    }  
}
```

**ALREADY THERE**

# 12 FACTOR APPS

Rewrite this to describe how Viper works in a lot of permutations.

**STRICT  
SEPARATION OF  
CONFIG FROM  
CODE**

CONFIG IS STUFF  
THAT VARIES IN  
DIFFERENT  
ENVIRONMENTS

CODE IS STUFF  
THAT STAYS THE  
SAME  
EVERYWHERE

THE 12 FACTOR  
APP STORES  
CONFIG IN ENV  
VARS

**READ  
FROM  
VIPER**

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command, args []string) {
    items, err := todo.ReadItems(dataFile))
    ...
    if err := todo.SaveItems(dataFile), items); err != nil {
```

# CMD/ADD.GO

```
func addRun(cmd *cobra.Command, args []string) {
    items, err := todo.ReadItems(viper.GetString("datafile"))

    ...
    if err := todo.SaveItems(viper.GetString("datafile"), items);
        err != nil {
```

# CMD/DONE.GO

```
func doneRun(cmd *cobra.Command, args []string) {  
    ...  
  
    items, err := todo.ReadItems(viper.GetString("datafile"))  
    ...  
  
    todo.SaveItems(viper.GetString("datafile"), items)
```

# CMD/LIST.GO

```
func listRun(cmd *cobra.Command, args []string) {  
    items, err := todo.ReadItems(viper.GetString("datafile"))  
    ...  
}
```

ENV

# USING DIFFERENT DATA FILE

```
>./tri list --datafile \
```

```
$HOME/Dropbox/Sync/tridos.json
```

```
>./tri add "Add config/ENV support" \
```

```
--datafile $HOME/Dropbox/Sync/tridos.json
```

# USING ENV

```
› DATAFILE=$HOME/Dropbox/Sync/trido.json \
./tri list

› DATAFILE=$HOME/Dropbox/Sync/trido.json \
./tri add "Add config/ENV support"
```

# EXPORT FTW

```
› export DATAFILE=$HOME/Dropbox/Sync/  
trido.json
```

```
› ./tri list
```

1. Add config/ENV support

2. Add config/ENV support

PLAYING  
NICE WITH  
OTHERS

# CMD/ROOT.GO

```
// Read in config file and ENV variables if set.  
func initConfig() {  
    viper.SetConfigName(".tri")  
    viper.AddConfigPath("$HOME")  
    viper.AutomaticEnv()  
viper.SetEnvPrefix("tri")  
  
    // If a config file is found, read it in.  
    if err := viper.ReadInConfig(); err == nil {  
        fmt.Println("Using config file:", viper.ConfigFileUsed())  
    }  
}
```

# EXPORT FTW

- › export **TRI\_DATAFILE**=\$HOME/Dropbox/  
Sync/trido.json
  - › ./tri list
1. Add config/ENV support
  2. Add config/ENV support

# CONFIG

# FILES

GOOD  
FOR APPS

# CREATE CONFIG FILE

```
> echo "datafile: /Users/  
spf13/Dropbox/Sync/  
trido.json" >  
$HOME/.tri.yaml
```

# USE CONFIG FILE

```
› go run main.go list
```

```
Using config file: /Users/  
spf13/.tri.yaml
```

1. Add config/ENV support
2. Add config/ENV support

# 11. WORKING AHEAD

**TRY TO  
IMPLEMENT  
EXTRA FEATURES**

# EDITING

# SEARCH

# CLOSING THOUGHTS

WE THINK GO  
IS AWESOME

**AWESOME  
COMMUNITY**

## Gophers

● Steve Francia

CHANNELS (439)

🔒 docker-gophercon

🔒 gc-lightning-2016

# general

🔒 go-market-growth

# go-miami

# golang-challenge

# golang-newbies

🔒 gopheracademy

# hugo

# showandtell

# vendor

# writing

DIRECT MESSAGES (10454) +

♥ slackbot

● Steve Francia (you)

● Aran Wilkinson



## #golang-newbies

2339 members | Welcome to #gola...



Search



intefaces. Then, when an obvious Today emerges or when you need to introduce a new type that behaves the same as the current one, extract the code and abstract into interfaces

This comes from the fact that interfaces are used to define behaviour that happens for certain pieces of data

For example: Basket interface{ Load(), Items(), Update() } would probably be something that I'd look into

09:42 ★ As it defines the behaviour of the Basket itself

And you don't need to define an interface for those either as you can have only one basket, so ideally the logic in the basket doesn't change

In the way of interaction

09:44 **Tom Fawssett** so in the example, the main bit I wanted a sanity check on is the way I've used a struct for dependency injection into the 'work' routine. the only reason I've used interfaces for the example is it saves the number of lines -> so I could start to write like this <https://gist.github.com/tom-f/82d375c894ba9e2e78925cdd7c8207f0>

09:44 **Florin Pătan** So probably I'd mock out the external things related to the basket And brb tunnel and work, catch up later

09:45 **Tom Fawssett** ApiCaller Call is accepting an interface so that I didn't have to



# ADDITIONAL RESOURCES

- Ashley's Learn to Code Resources
- Steve's Blog
- Go Girls Who Code

THANK  
YOU