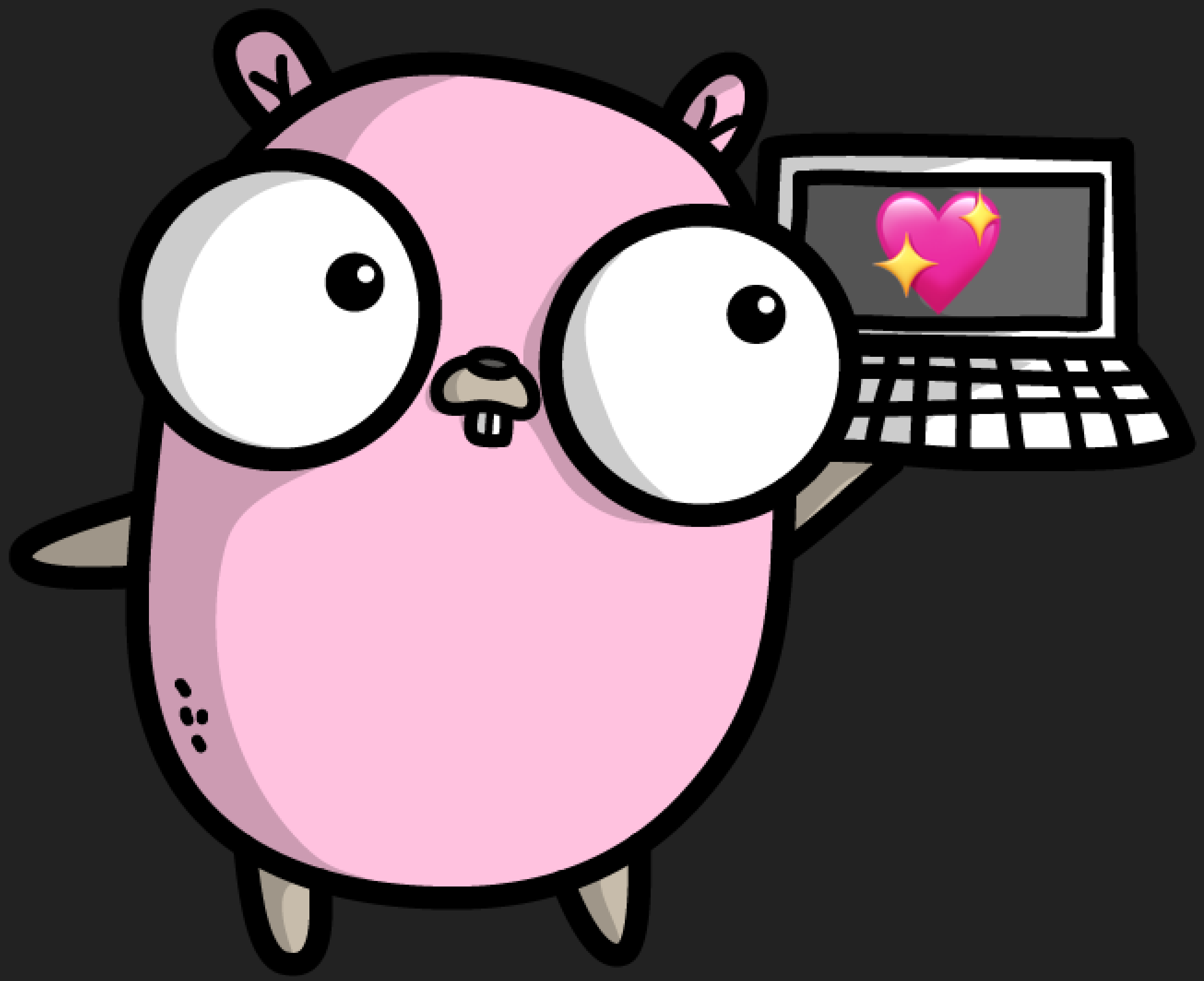
Designing Command-Line Tools People Love



Carolyn Van Slyck
Senior Software Engineer at Microsoft

Often CLIs aren't designed,
functionality is added haphazardly

Design Goals

- Predictable
- Task oriented
- Friendly to both people and scripts
- High Quality

My CLI CV

Docker Version Manager

dep

Service Catalog

porter

Command Design

Pick your grammar

A system of rules that defines the structure of a command

Understand precedent in your ecosystem

- service catalog followed kubectl
- docker version manager followed node version manager
- dep didn't follow glide
- porter is setting precedent

Let's design a CLI!

¯_(ツ)_/¯

Commands that read like sentences are easier to remember

```
$ emote add emoticon gopher --value ʕ •ᴥ•ʔ
added custom emoticon "gopher"

$ emote delete emoticon anxious
deleted custom emoticon "anxious"
```

Avoid positional arguments where the order matters

```
$ emote add repo funk https://x.com/funk.json 🤔

$ emote add repo https://x.com/funk.json funk 😅

$ emote add repo funk --url https://x.com/funk.json ✅
$ emote repo delete funk moar-funk

deleted funk and moar-funk
```

Support automation on your commands

```
$ emote list repos --output json
[
  {
    "name":"funk",
    "url":"https://example.com/funk.json",
  "size":100,
  "created":"2019-07-15T14:32:22Z"
  }
]
```

Default to human first output

I can't read ISO

Sometimes the resource is implicit in the domain

```
$ emote list
NAME                VALUE
shrug               ¯\_(ツ)_/¯
tableflip           (╯°□°）╯︵ ┻━┻
monocle             ಠ_ರೃ
```

Aliases provide balance between brevity and discoverability

```
$ emote emoticon list
NAME                VALUE
shrug               ¯\_(ツ)_/¯
tableflip           (╯°□°)╯︵ ┻━┻
monocle             ಠ_ರೃ
```

Customize your help text

```
$ emote --help
emote helps you react in realtime

Resources:
  emoticons
  repos

Aliased Commands:
  list    List emoticons
```

Task-oriented commands are the most helpful

```
$ emote shrug
¯\_(ツ)_/¯ copied to the clipboard


$ emote shrug --dest slack
Your slack status is now ¯\_(ツ)_/¯
```

Domain vs. Grammar

Use your judgement about the domain when breaking with the grammar

Combine commands to make tasks easier

```
$ travis encrypt MY_SECRET_ENV=super_secret --add env
```

1. Download the public key for your travis repository
2. Encrypt the env var with the public key using openssl
3. Insert an entry into .travis.yml with the encrypted value

```
$ travis pubkey | jq -r .key > mykey.pub
$ echo 'MY_SECRET_ENV=super_secret' \
  | openssl rsautl -encrypt -pubin -inkey mykey.pub \
  | travis env add
```

Piping is good for automation but people don't want to pipe

Give people a single command to perform a task and they will thank you

Progress Towards Our Goal:
CLI People ❤️

✅ Easy to learn and remember
✅ Solves day-to-day tasks
❓ High Quality

A great CLI needs to have high quality code backing it

Let's build a CLI!

```
$ emote shrug
¯\_(ツ)_/¯
```

The final code is available at github.com/carolynvs/emote

What is spf13/cobra

- CLI Framework / Main Entrypoint
- Command Routing
- Error Handling
- Help Text
- Flag Parsing and Validation

Emote CLI Wiring

```go
package main

import (
  "fmt"
  "os"

  "github.com/atotto/clipboard"
  "github.com/spf13/cobra"
)

func main() {
  cmd := buildEmoteCommand()
  if err := cmd.Execute(); err != nil {
    os.Exit(1)
  }
}

func buildEmoteCommand() *cobra.Command {
  emote := &cobra.Command{
    Use:    "emote",
  }
  emote.AddCommand(buildShrugCommand())
  return emote
}
```

Shrug Wiring

```go
func buildShrugCommand() *cobra.Command {
  var dest string

  shrug := &cobra.Command{
    Use: "shrug",
    Run: func(cmd *cobra.Command, args []string) {
      const emoticon = `¯\_(ツ)_/¯`
      switch dest {
      case "clipboard":
        clipboard.WriteAll(emoticon)
        fmt.Println(emoticon, "was copied to the clipboard")
      default:
        fmt.Println(emoticon)
      }
    },
  }

  shrug.Flags().StringVar(&dest, "dest", "clipboard", "Where to send your emoticon")

  return shrug
}
```

Pro Tip: Create an Application Package

- Make functions that correspond 1:1 to the commands in your CLI
- Create happy little packages for everything
- Forget this is a CLI and follow your dreams 🌈

Emoticons Application Package

```go
package emoticons

import (
  "fmt"
  "github.com/atotto/clipboard"
)

type App struct {}

func (a *App) Shrug(dest string) {
  const emoticon = `¯\_(ツ)_/¯`

  switch dest {

  case "clipboard":
    clipboard.WriteAll(emoticon)
    fmt.Println(emoticon, "was copied to the clipboard")

  default:
    fmt.Println(emoticon)
  }
}
```

Shrug Wiring with Application

```go
shrug := &cobra.Command{
  Use: "shrug",
  Run: func(cmd *cobra.Command, args []string) {

    // Much Better! 👍
    app := emoticons.App{}
    app.Shrug(dest)

  },
}
```

Let's Add Configuration

```
dest = "slack"

[emoticon]
  shrug = '¯\_(ツ)_/¯'
  tableflip = '(╯°□°) ╯︵ ┻━┻'
```

But I like yaml better

```
dest: "slack"

emoticon:
  shrug: '¯\_(ツ)_/¯'
  tableflip: '(╯°□°) ╯︵ ┻━┻'
```

Excuse me, I need json for reasons...

```
{
  "dest": "slack",
  "emoticon": {
    "shrug": "¯\_(ツ)_/¯",
    "tableflip": "(╯°□°) ╯︵ ┻━┻"
  }
}
```

Meet your users where they are

Why Not Both?

What is spf13/viper

- Single combined configuration from multiple sources
- Reads from flags, config files, remote key/value stores, environment variables
- Smart defaulting: can tell if it was defaulted or set by the user
- Supports config files of multiple formats: json, yaml, toml, and more

Application Package with Viper

```
import "github.com/spf13/viper"

type App struct {
  viper *viper.Viper
}

func New() (*App, error) {
  v := viper.New()
  v.AddConfigPath(".")

  err := v.ReadInConfig()
  if err != nil {
    return nil, err
  }
  return &App{viper: v}, nil
}

func (a *App) Emote(name string, dest string) {
  emoticon := a.viper.GetString("emoticon." + name)

  ...
}
```

Emote CLI Wiring with Dynamic Commands

```go
func buildEmoteCommand() *cobra.Command {
  app, err := emoticons.New()
  if err != nil {
    log.Fatal(err)
  }
  var dest string

  emote := &cobra.Command{
    Use:    "emote",
    Run: func(cmd *cobra.Command, args []string) {
      emoticonName := args[0]
      app.Emote(emoticonName, dest)
    },
    Args: cobra.ExactArgs(1),
  }
...
```

Pro Tip: Keep Viper Isolated

Config Package

```go
package config

import (
  "github.com/spf13/viper"
)

type Config struct {
  Emoticon map[string]string
}

func Load() (*Config, error) {
  v := viper.New()
  v.AddConfigPath(".")

  err := v.ReadInConfig()
  if err != nil {
    return nil, err
  }

  c := &Config{}
  err = v.Unmarshal(c)
  return c, err
}
```

Application with Viper Tucked Away

```go
type App struct {
  Config *config.Config
}

func New() (*App, error) {
  c, err := config.Load()
  if err != nil {
    return nil, err
  }
  return &App{Config: c}, nil
}

func (a *App) Emote(name string, dest string) {
  // Yay! This feels more intuitive 👍
  emoticon := a.Config.Emoticon[name]
```

Default Flags with Viper

```
type Config struct {
  Dest string // Set by --dest or config file if not present
  Emoticon map[string]string
}

func (c *Config) Load(cmd *cobra.Command) error {
  ...

  // Bind the cobra flags to our config file
  v.BindPFlags(cmd.Flags())

  ...
}
```

Final Emote Wiring

```go
func buildEmoteCommand() *cobra.Command {
  app := emoticons.New()

  emote := &cobra.Command{
    Use: "emote",
    PreRunE: func(cmd *cobra.Command, args []string) error {
      return app.Config.Load(cmd)
    },
    Run: func(cmd *cobra.Command, args []string) {
      emoticonName := args[0]
      app.Emote(emoticonName)
    },
    Args: cobra.ExactArgs(1),
  }

  emote.Flags().StringVar(&app.Config.Dest, "dest", "clipboard", "Where to send your emoticon")

  return emote
}
```

Testing Emote

```go
package emoticons

import (
  "bytes"
  "testing"

  "github.com/carolynvs/emote/config"
  "github.com/stretchr/testify/assert"
)

func TestApp_Emote(t *testing.T) {
  const shrugEmoticon = `¯\_(ツ)_/¯`

  out := &bytes.Buffer{}
  app := &App{
    Out: out,
    Config: &config.Config{
      Emoticon: map[string]string{"shrug": shrugEmoticon},
    },
  }

  app.Emote("shrug")

  assert.Contains(t, out.String(), shrugEmoticon)
}
```

We did it!

Gopher artwork by Ashley McNamara
licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License

References

- `github.com/carolynvs/emote`
- `github.com/spf13/cobra - Commands and Flags`
- `github.com/spf13/viper - Configuration Management`
- `github.com/spf13/afero - File System Abstraction`
- `github.com/dustin/go-humanize - Natural Language Units`

Thank you, Steve Francia! 💖