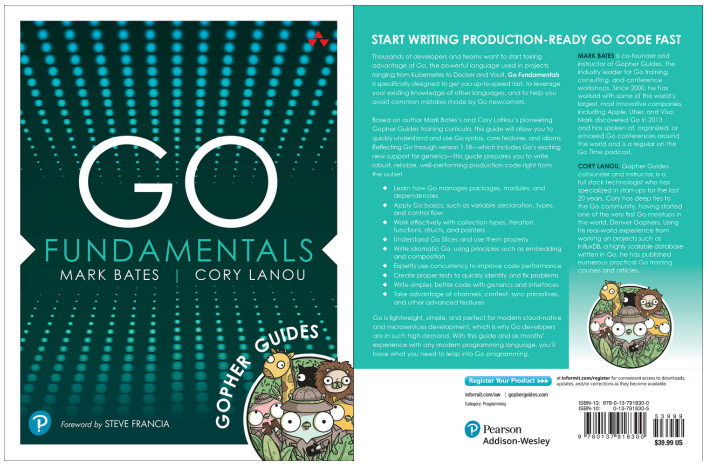
Building Better CLIs

Go Fundamentals



Building Better CLIs

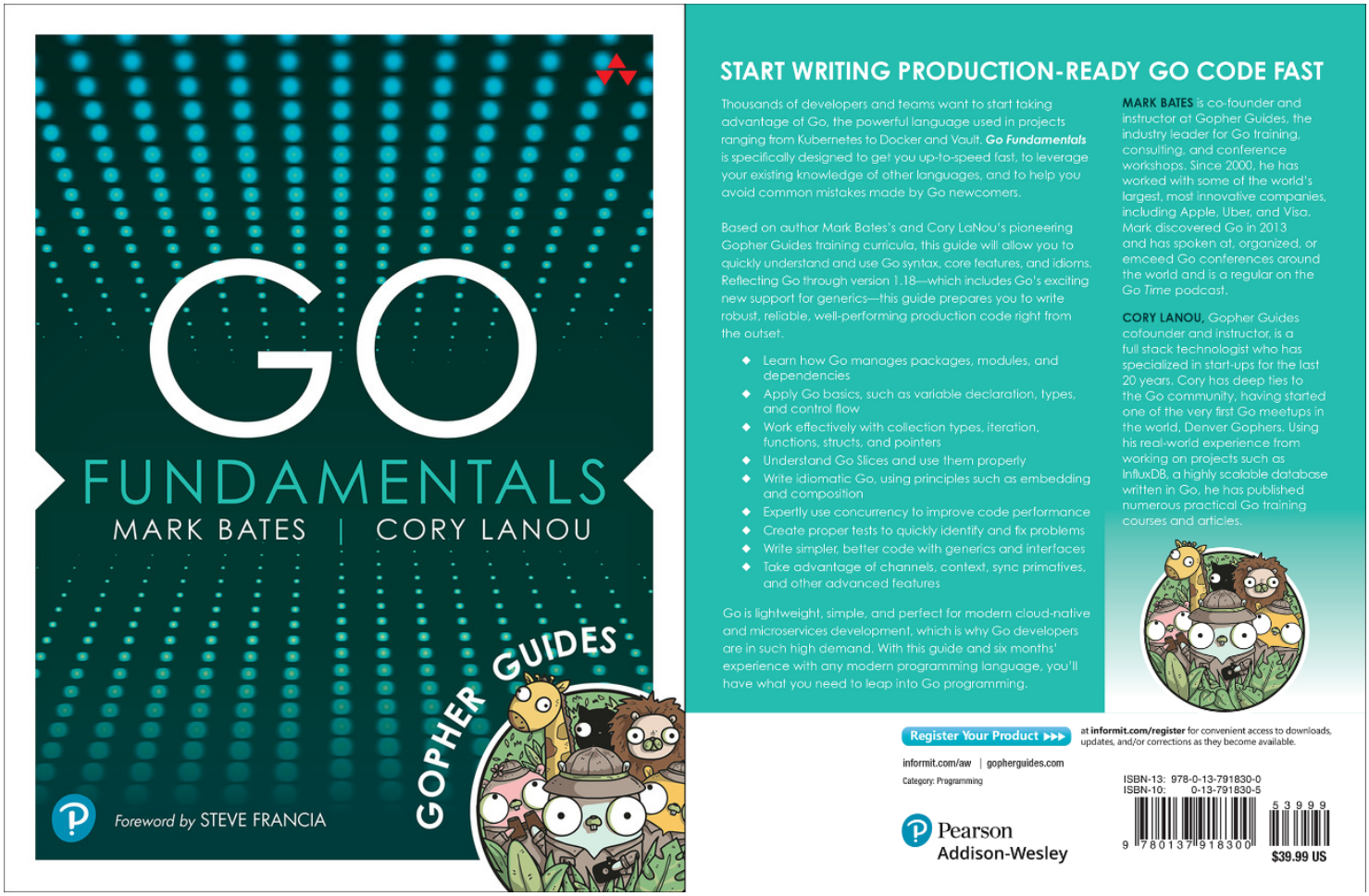
- Avoiding Globals
- Cleaner Code
- Easily Testable
- Composable CLI Apps
- Framework/3rd Party Free!*

The Library

```
$ go doc github.com/markbate

package bostongo // import "

type Walker struct {
        PrintDirs bool
        SkipFiles bool
}

func (wk Walker) Walk(cab fs

----------------------------
Go Version: go1.21.0
```

Figure 1.1: walker.go

Four Commands



Building Better CLIs

- Avoiding Globals
- Cleaner Code
- Easily Testable
- Composable CLI Apps
- Framework/3rd Party Free!*

The Library

```
$ go doc github.com/markbates/bostongo.Walker

package bostongo // import "github.com/markbates/bostongo"

type Walker struct {
        PrintDirs bool
        SkipFiles bool
}

func (wk Walker) Walk(cab fs.FS, w io.Writer) error

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.1: walker.go

Four Commands

- `walker` - simple CLI app
- `server` - an HTML server
- `bostongo` - a combined CLI app
  - `garlic` - generates a garlic app (*bonus*)

Globals

Command Line Arguments

```
$ go doc os.Args

package os // import "os"

var Args []string
    Args hold the command-line arguments, starting with the program name.

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.2: `os.Args`

Current Working Directory

```
$ go doc os.Getwd

package os // import "os"

func Getwd() (dir string, err error)
    Getwd returns a rooted path name corresponding to the current directory.
    If the current directory can be reached via multiple paths (due to symbolic
    links), Getwd may return any one of them.

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.3: `os.Getwd`

I/O

```
$ go doc os.Stdout

package os // import "os"

var (
        Stdin  = NewFile(uintptr(syscall.Stdin), "/dev/stdin")
        Stdout = NewFile(uintptr(syscall.Stdout), "/dev/stdout")
        Stderr = NewFile(uintptr(syscall.Stderr), "/dev/stderr")
)
    Stdin, Stdout, and Stderr are open Files pointing to the standard input,
    standard output, and standard error file descriptors.

    Note that the Go runtime writes to standard error for panics and crashes;
    closing Stderr may cause those messages to go elsewhere, perhaps to a file
    opened later.

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.4: `os.Stdout`

File System

```
$ go doc os.Open

package os // import "os"

func Open(name string) (*File, error)
    Open opens the named file for reading. If successful, methods on the
    returned file can be used for reading; the associated file descriptor has
    mode O_RDONLY. If there is an error, it will be of type *PathError.

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.5: `os.Open`

```
$ go doc os.Stat

package os // import "os"

func Stat(name string) (FileInfo, error)
    Stat returns a FileInfo describing the named file. If there is an error,
    it will be of type *PathError.

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.6: `os.Stat`

Environment Variables

```
$ go doc os.Getenv

package os // import "os"

func Getenv(key string) string
    Getenv retrieves the value of the environment variable named by the key.
    It returns the value, which will be empty if the variable is not present.
    To distinguish between an empty value and an unset value, use LookupEnv.

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.7: `os.Getenv`

The `main` Function

```go
func main() {}
```

Figure 1.8: Example `main` function.

A Simple CLI

The `main` Function

```go
func main() {
    args := os.Args[1:]

    pwd, err := os.Getwd()
    if err != nil {
        log.Fatal(err)
    }

    ctx := context.Background()
    ctx, cancel := signal.NotifyContext(ctx, os.Interrupt)
    defer cancel()

    app := cli.App{}
    err = app.Main(ctx, pwd, args)

    if err != nil {
        log.Fatal(err)
    }
}
```

Figure 1.9: cmd/walker/main.go

Information Gathered

- Command Line Arguments
- Current Working Directory
- Context

The Imports

```go
import (
    "context"
    "log"
    "os"
    "os/signal"

    "github.com/markbates/bostongo/cmd/walker/cli"
)
```

Figure 1.10: cmd/walker/main.go

Directory Tree

```
$ tree cmd/walker -I testdata

cmd/walker
|-- cli
|   |-- app.go
|   `-- app_test.go
`-- main.go

2 directories, 3 files
```

Figure 1.11: Directory structure of the `cmd/walker` command.

The `cli.App` Type

```
$ go doc ./cmd/walker/cli.App

package cli // import "github.com/markbates/bostongo/cmd/walker/cli"

type App struct {
        // IO to be used by the app
        iox.IO

        // Cab is the file system to walk. Defaults to os.DirFS(pwd).
        Cab fs.FS

        // Has unexported fields.
}
    App is the main application struct for the walker command.

func (a *App) Describe() string
func (a *App) Main(ctx context.Context, pwd string, args []string) error
func (a *App) Print(w io.Writer) error
func (a *App) SetIO(oi iox.IO)

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.12: cmd/walker/cli/app.go

Standard I/O

```
$ go doc os.Stdout

package os // import "os"

var (
        Stdin  = NewFile(uintptr(syscall.Stdin), "/dev/stdin")
        Stdout = NewFile(uintptr(syscall.Stdout), "/dev/stdout")
        Stderr = NewFile(uintptr(syscall.Stderr), "/dev/stderr")
)
    Stdin, Stdout, and Stderr are open Files pointing to the standard input,
    standard output, and standard error file descriptors.

    Note that the Go runtime writes to standard error for panics and crashes;
    closing Stderr may cause those messages to go elsewhere, perhaps to a file
    opened later.

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.13: `os.Stdout`

The `IO` Type

```
$ go doc github.com/markbates/iox.IO

package iox // import "github.com/markbates/iox"

type IO struct {
        In  io.Reader `json:"-"` // standard input
        Out io.Writer `json:"-"` // standard output
        Err io.Writer `json:"-"` // standard error
}
    IO represents the standard input, output, and error stream.

func Discard() IO
func (oi IO) Stderr() io.Writer
func (oi IO) Stdin() io.Reader
func (oi IO) Stdout() io.Writer

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.14: `iox.IO`

The `Stdout` Method

```go
// Stdout returns IO.In.
// Defaults to os.Stdout.
func (oi IO) Stdout() io.Writer {
    if oi.Out == nil {
        return os.Stdout
    }

    return oi.Out
}
```

Figure 1.15: `iox.IO.Stdout`

In Testing

```go
bb := &strings.Builder{}
oi := iox.IO{
    Out: bb,         // use the strings.Builder as STDOUT
    Err: io.Discard, // discard STDERR
}
```

Figure 1.16: cmd/bostongo/cli/garlic_test.go

The `cli.App` Type

```
$ go doc ./cmd/walker/cli.App

package cli // import "github.com/markbates/bostongo/cmd/walker/cli"

type App struct {
        // IO to be used by the app
        iox.IO

        // Cab is the file system to walk. Defaults to os.DirFS(pwd).
        Cab fs.FS

        // Has unexported fields.
}
    App is the main application struct for the walker command.

func (a *App) Describe() string
func (a *App) Main(ctx context.Context, pwd string, args []string) error
func (a *App) Print(w io.Writer) error
func (a *App) SetIO(oi iox.IO)

------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.17: cmd/walker/cli/app.go

The `fs.FS` Interface

```
$ go doc io/fs.FS

package fs // import "io/fs"

type FS interface {
        // Open opens the named file.
        //
        // When Open returns an error, it should be of type *PathError
        // with the Op field set to "open", the Path field set to name,
        // and the Err field describing the problem.
        //
        // Open should reject attempts to open names that do not satisfy
        // ValidPath(name), returning a *PathError with Err set to
        // ErrInvalid or ErrNotExist.
        Open(name string) (File, error)
}
    An FS provides access to a hierarchical file system.

    The FS interface is the minimum implementation required of the file system.
    A file system may implement additional interfaces, such as ReadFileFS,
    to provide additional or optimized functionality.

func Sub(fsys FS, dir string) (FS, error)

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.18: `fs.FS`

The `Commander` Interface

```go
type Commander interface {
    Main(ctx context.Context, pwd string, args []string) error
}
```

Figure 1.19: cmd/bostongo/cli/ifaces.go

Default File System

```
$ go doc os.DirFS

package os // import "os"

func DirFS(dir string) fs.FS
    DirFS returns a file system (an fs.FS) for the tree of files rooted at the
    directory dir.

    Note that DirFS("/prefix") only guarantees that the Open calls
    it makes to the operating system will begin with "/prefix":
    DirFS("/prefix").Open("file") is the same as os.Open("/prefix/file").
    So if /prefix/file is a symbolic link pointing outside the /prefix tree,
    then using DirFS does not stop the access any more than using os.Open does.
    Additionally, the root of the fs.FS returned for a relative path,
    DirFS("prefix"), will be affected by later calls to Chdir. DirFS is
    therefore not a general substitute for a chroot-style security mechanism
    when the directory tree contains arbitrary content.

    The directory dir must not be "".

    The result implements io/fs.StatFS, io/fs.ReadFileFS and io/fs.ReadDirFS.

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.20: `os.DirFS`

The `cli.App#Main` Method

```go
// Main is the main entry point for the walker command.
func (a *App) Main(ctx context.Context, pwd string, args []string) error {
    if a == nil {
        return fmt.Errorf("nil app")
    }

    if ctx == nil {
        ctx = context.Background()
    }

    wk := bostongo.Walker{}
    flags := a.flags(&wk)

    err := flags.Parse(args)
    if err != nil {
        return err

    }

    a.mu.RLock()

    cab := a.Cab
    oi := a.IO

    a.mu.RUnlock()

    args = flags.Args()
    if len(args) > 0 {
        pwd = args[0]
    }

    if cab == nil {
        cab = os.DirFS(pwd)
    }

    sctx, cause := context.WithCancelCause(ctx)
    defer cause(nil)

    // launch as a goroutine so if it takes too
    // long we can cancel the command.
    go func() {
        err := wk.Walk(cab, oi.Stdout())
        cause(err)
    }()

    <-sctx.Done()

    err = context.Cause(sctx)
    if err != nil && err != context.Canceled {
        return err
    }

    return nil
}
```

Figure 1.21: cmd/walker/cli/app.go

The `cli.App#flags` Method

```go
func (a *App) flags(wk *bostongo.Walker) *flag.FlagSet {

    flags := flag.NewFlagSet("walker", flag.ContinueOnError)

    flags.SetOutput(a.Stderr())
    flags.BoolVar(&wk.PrintDirs, "dirs", false, "print directories")
    flags.BoolVar(&wk.SkipFiles, "skip-files", false, "skip files")

    return flags
}
```

Figure 1.22: cmd/walker/cli/app.go

Running the `walker` Command

```
$ go run cmd/walker/main.go -h

Usage of walker:
  -dirs
        print directories
  -skip-files
        skip files
2023/08/29 15:14:09 flag: help requested
exit status 1

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.23: The `walker` command help output.

```
$ go run cmd/walker/main.go -dirs testdata

a
a/a.md
a/b
a/b/b.md
a/b/c
a/b/c/c.md

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.24: Running the `walker` command.

Testing the `walker` Command

```go
t.Run(tc.name, func(t *testing.T) {
    r := require.New(t)

    bb := &bytes.Buffer{}

    app := tc.app

    if app != nil {
        app.IO.Out = bb
    }

    ctx := context.Background()
    ctx, cause := context.WithTimeout(ctx, time.Second)
    defer cause()

    err := app.Main(ctx, root, tc.args)

    if tc.err {
        r.Error(err)
        return
    }

    r.NoError(err)

    r.Equal(tc.exp, bb.String())
})
```

Figure 1.25: cmd/walker/cli/app_test.go

```
$ go test -v

=== RUN   Test_App_Main
=== PAUSE Test_App_Main
=== CONT  Test_App_Main
=== RUN   Test_App_Main/files_only/with_cab
=== RUN   Test_App_Main/files_only/without_cab
=== RUN   Test_App_Main/dirs_only
=== RUN   Test_App_Main/all
=== RUN   Test_App_Main/nil_app
--- PASS: Test_App_Main (0.00s)
    --- PASS: Test_App_Main/files_only/with_cab (0.00s)
    --- PASS: Test_App_Main/files_only/without_cab (0.00s)
    --- PASS: Test_App_Main/dirs_only (0.00s)
    --- PASS: Test_App_Main/all (0.00s)
    --- PASS: Test_App_Main/nil_app (0.00s)
PASS
ok      github.com/markbates/bostongo/cmd/walker/cli    0.087s

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.26: Running the `walker` command tests.

Globals Avoided!

- ~~Command Line Arguments~~
- ~~Current Working Directory~~
- ~~I/O~~
- ~~File System~~
- Environment Variables

The `web` Library

```
$ go doc github.com/markbates/bostongo/web.App

package web // import "github.com/markbates/bostongo/web"

type App struct {
        DefaultPath string
}

func (a App) ServeHTTP(w http.ResponseWriter, r *http.Request)

------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.27: cmd/server/cli/app.go

The `ServeHTTP` ` Method

```go
func (a App) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    if r == nil {
        http.Error(w, "nil request", http.StatusBadRequest)
        return
    }

    f, err := a.parseForm(r)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    if err := a.walk(&f); err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    tmpl, err := template.New("").Parse(htmlTemplate)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    err = tmpl.Execute(w, f)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

}
```

Figure 1.28: web/app.go

The `walk` Method

```go
func (a App) walk(f *form) error {
    wk := bostongo.Walker{
        PrintDirs: len(f.PrintDirs) > 0,
        SkipFiles: len(f.SkipFiles) > 0,
    }
    bb := &bytes.Buffer{}

    cab := os.DirFS(f.WalkPath)
    if err := wk.Walk(cab, bb); err != nil {
        return err
    }

    f.Results = bb.String()
    return nil
}
```

Figure 1.29: web/app.go

The Server CLI

Directory Tree

```
$ tree cmd/server -I testdata

cmd/server
|-- cli
|   |-- app.go
|   |-- app_test.go
|   |-- env.go
|   `-- env_test.go
`-- main.go

2 directories, 5 files
```

Figure 1.30: Directory structure of the `cmd/server` command.

The `main` Function

```go
func main() {
    args := os.Args[1:]

    pwd, err := os.Getwd()
    if err != nil {
        log.Fatal(err)
    }

    ctx := context.Background()
    ctx, cancel := signal.NotifyContext(ctx, os.Interrupt)
    defer cancel()

    app := cli.App{}
    err = app.Main(ctx, pwd, args)
    if err != nil {
        log.Fatal(err)
    }

    <-app.Done()
}
```

Figure 1.31: cmd/server/main.go

The Server `cli.App` Type

```
$ go doc ./cmd/server/cli.App

package cli // import "github.com/markbates/bostongo/cmd/server/cli"

type App struct {
        // IO to be used by the app
        iox.IO

        // Web app to be used by the app
        web.App

        // Server to be used by the app
        // If nil, a default server will be created.
        Server *http.Server

        // Port to listen on. Defaults to 3000.
        Port int

        // Env to be used by the app
        // If nil, os.Getenv will be used.
        *Env

        // Has unexported fields.
}

func (a *App) Describe() string
func (a *App) Done() <-chan struct{}
func (a *App) Getenv(key string) (s string)
func (a *App) Main(ctx context.Context, pwd string, args []string) error
func (a *App) Print(w io.Writer) error
func (a *App) SetIO(oi iox.IO)

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.32: cmd/server/cli/app.go

The `Env` Type

```go
type Env struct {
    data map[string]string
    mu   sync.RWMutex
}

func (e *Env) Getenv(key string) string
func (e *Env) Setenv(key string, value string)
```

Figure 1.33: cmd/server/cli/env.go

The `Getenv` Method

```go
func (e *Env) Getenv(key string) string {
    if e == nil || e.data == nil {
        return os.Getenv(key)
    }

    e.mu.RLock()
    defer e.mu.RUnlock()

    if k, ok := e.data[key]; ok {
        return k
    }

    return os.Getenv(key)
}
```

Figure 1.34: cmd/server/cli/env.go

Using the `Env` Type

```go
if port == 0 {
    p := a.Getenv("PORT")
    pi, _ := strconv.Atoi(p)
    if pi == 0 {
        pi = 3000
    }
    port = pi
}
```

Figure 1.35: cmd/server/cli/app.go

The `cli.App#Main` Function

```go
func (a *App) Main(ctx context.Context, pwd string, args []string) error {
    if a == nil {
        return fmt.Errorf("nil app")
    }

    flags := a.flags()
    err := flags.Parse(args)
    if err != nil {
        return err
    }

    srv, err := a.server()
    if err != nil {
        return err
    }

    sctx, cause := context.WithCancelCause(ctx)
    defer cause(nil)

    srv.BaseContext = func(_ net.Listener) context.Context {
        return sctx
    }

    go func() {
        <-ctx.Done()

        ctx, cancel := context.WithTimeout(sctx, 2*time.Second)
        defer cancel()

        cause(srv.Shutdown(ctx))

        a.mu.Lock()
        defer a.mu.Unlock()

        a.once.Do(func() {
            if a.quit != nil {
                close(a.quit)
            }
            a.quit = nil
        })

    }()

    if err := srv.ListenAndServe(); err != nil {
        cause(err)
    }

    err = context.Cause(sctx)
    if err != nil && err != context.Canceled {
        return err
    }

    return nil
}
```

Figure 1.36: cmd/server/cli/app.go

Testing the `server` Command

```
$ go test -v

=== RUN   Test_App_Main
=== PAUSE Test_App_Main
=== RUN   Test_App_Getenv
=== PAUSE Test_App_Getenv
=== RUN   Test_Env_Getenv
=== PAUSE Test_Env_Getenv
=== CONT  Test_App_Main
=== CONT  Test_Env_Getenv
=== CONT  Test_App_Getenv
=== RUN   Test_App_Getenv/default
--- PASS: Test_Env_Getenv (0.00s)
=== RUN   Test_App_Getenv/missing_key
=== RUN   Test_App_Getenv/good_key
--- PASS: Test_App_Getenv (0.00s)
    --- PASS: Test_App_Getenv/default (0.00s)
    --- PASS: Test_App_Getenv/missing_key (0.00s)
    --- PASS: Test_App_Getenv/good_key (0.00s)
--- PASS: Test_App_Main (0.00s)
PASS
ok      github.com/markbates/bostongo/cmd/server/cli    0.197s

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.37: Running the `server` command tests.

Globals Avoided!

- ~~Command Line Arguments~~
- ~~Current Working Directory~~
- ~~I/O~~
- ~~File System~~
- ~~Environment Variables~~

Combining Commands

Directory Tree

```
$ tree cmd/bostongo -I testdata

cmd/bostongo
|-- cli
|   |-- app.go
|   |-- app_test.go
|   |-- commands.go
|   |-- garlic.go
|   |-- garlic_test.go
|   `-- ifaces.go
`-- main.go

2 directories, 7 files
```

Figure 1.38: Directory structure of the `cmd/bostongo` command.

The `cli.App` Type

```
$ go doc ./cmd/bostongo/cli.App

package cli // import "github.com/markbates/bostongo/cmd/bostongo/cli"

type App struct {
        iox.IO

        Commands *Commands

        // Has unexported fields.
}

func (a *App) Main(ctx context.Context, pwd string, args []string) error
func (*App) PluginName() string
func (a *App) Print(w io.Writer) error
func (a *App) SetIO(io iox.IO)

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.39: cmd/bostongo/cli/app.go

The `cli.Commands` Type

```
$ go doc ./cmd/bostongo/cli.Commands

package cli // import "github.com/markbates/bostongo/cmd/bostongo/cli"

type Commands struct {
        // Has unexported fields.
}

func (c *Commands) Find(name string) (Commander, bool)
func (c *Commands) Map() map[string]Commander
func (c *Commands) Print(w io.Writer) error
func (c *Commands) Set(name string, cmd Commander)

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.40: cmd/bostongo/cli/app.go

The `cli.Commands#Find` Method

```go
func (c *Commands) Find(name string) (Commander, bool) {
    if c == nil {
        return nil, false
    }

    c.mu.RLock()
    defer c.mu.RUnlock()

    cmd, ok := c.routes[name]
    if !ok || cmd == nil {
        return nil, false
    }

    return cmd, true
}
```

Figure 1.41: cmd/bostongo/cli/app.go

The `Commander` Interface

```go
type Commander interface {
    Main(ctx context.Context, pwd string, args []string) error
}
```

Figure 1.42: cmd/bostongo/cli/ifaces.go

The `cli.App#Main` Method

```go
func (a *App) Main(ctx context.Context, pwd string, args []string) error {
    if a == nil {
        return fmt.Errorf("nil app")
    }

    flags := a.flags()
    err := flags.Parse(args)
    if err != nil {
        return err
    }

    cmds, err := a.populateCommands()
    if err != nil {
        return err
    }

    args = flags.Args()

    if len(args) == 0 {
        if e := a.Print(a.Stderr()); e != nil {
            return e
        }
        return fmt.Errorf("no command given")
    }

    // snippet: work
    cmd, ok := cmds.Find(args[0])
    if !ok {
        if e := a.Print(a.Stderr()); e != nil {
            return e
        }
        return fmt.Errorf("unknown command %q", args[0])
    }

    if sio, ok := cmd.(SettableIO); ok {
        sio.SetIO(a.IO)
    }

    return cmd.Main(ctx, pwd, args[1:])
    // snippet: work
}
```

Figure 1.43: cmd/bostongo/cli/app.go

The SettableIO Interface

```go
type SettableIO interface {
    SetIO(io iox.IO)
}
```

Figure 1.44: cmd/bostongo/cli/ifaces.go

Running the `bostongo` Command

```
$ go run cmd/bostongo/main.go walker -dirs testdata

a
a/a.md
a/b
a/b/b.md
a/b/c
a/b/c/c.md

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.45: Running the `bostongo` command.

Testing the `bostongo` Command

```go
func Test_App_Main(t *testing.T) {
    t.Parallel()

    r := require.New(t)

    bb := &bytes.Buffer{}
    oi := iox.IO{
        Out: bb,
    }

    app := &App{
        IO: oi,
    }

    ctx := context.Background()

    err := app.Main(ctx, "testdata", []string{"walker"})
    r.NoError(err)

    // assert the output
    exp := `a/a.md
a/b/b.md
a/b/c/c.md`

    act := bb.String()
    act = strings.TrimSpace(act)

    r.Equal(exp, act)
}
```

Figure 1.46: cmd/bostongo/cli/app_test.go

```
$ go test -v -run App

=== RUN   Test_App_Main
=== PAUSE Test_App_Main
=== CONT  Test_App_Main
--- PASS: Test_App_Main (0.00s)
PASS
ok      github.com/markbates/bostongo/cmd/bostongo/cli  0.340s

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.47: Running the `bostongo` command tests.

The Garlic Pattern

The Problem

- CLI toolchain versioning
- Extending CLI toolchains

The Solution

- User runs `<command x>` in their project
- Look for a local version of `<command x>`
- If found, shell out to local version
- If not found continue using the `<command x>` binary

The `garlic.Garlic` Type

```
$ go doc github.com/markbates/garlic.Garlic

package garlic // import "github.com/markbates/garlic"

type Garlic struct {
        Cmd  Commander
        FS   fs.FS
        IO   iox.IO
        Name string
}

func (g *Garlic) Main(ctx context.Context, pwd string, args []string) error
func (g Garlic) PluginName() string

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.48: `garlic.Garlic`

Garlic Commander

```go
type Commander interface {
    Main(ctx context.Context, pwd string, args []string) error
}
```

Figure 1.49: `garlic.Commander`

The `garlic.Garlic#Main` Method

```go
func (g *Garlic) Main(ctx context.Context, pwd string, args []string) error {
    if g == nil {
        return fmt.Errorf("garlic is nil")
    }

    if len(g.Name) == 0 {
        return fmt.Errorf("command name is required")
    }

    local := Local{
        FS:   g.FS,
        IO:   g.IO,
        Name: g.Name,
        Root: pwd,
    }

    if local.Exists() {
        return local.Run(ctx, args)
    }

    if g.Cmd == nil {
        return fmt.Errorf("command is nil")
    }

    cmd := g.Cmd

    if sfs, ok := cmd.(SettableFS); ok {
        sfs.SetFS(g.FS)
    }

    if sio, ok := cmd.(SettableIO); ok {
        sio.SetIO(g.IO)
    }

    return cmd.Main(ctx, pwd, args)
}
```

Figure 1.50: `garlic.Garlic.Main`

The `main` Function

```go
func main() {
    args := os.Args[1:]

    pwd, err := os.Getwd()
    if err != nil {
        log.Fatal(err)
    }

    ctx, cancel := signal.NotifyContext(context.Background(), os.Interrupt)
    defer cancel()

    app := &cli.App{}

    clove := &garlic.Garlic{
        Name: app.PluginName(),
        Cmd:  app,
        FS:   os.DirFS(pwd),
    }

    err = clove.Main(ctx, pwd, args)
    if err != nil {
        log.Fatal(err)
    }
}
```

Figure 1.51: cmd/bostongo/main.go

Testing Garlic

```go
func Test_Garlic_Works(t *testing.T) {
    t.Parallel()
    r := require.New(t)

    // generate the garlic directory
    // and the main.go file
    _, dir := garlicDir(t)

    bb := &strings.Builder{}
    oi := iox.IO{
        Out: bb,         // use the strings.Builder as STDOUT
        Err: io.Discard, // discard STDERR
    }

    clove := &garlic.Garlic{
        Cmd:  &App{},        // the App to run, if no local command is found
        FS:   os.DirFS(dir), // the filesystem to use
        IO:   oi,            // IO to be used
        Name: "bostongo",    // the name of the command to run
    }

    // call the walker command through garlic
    err := clove.Main(context.Background(), dir, []string{"walker"})
    r.NoError(err)

    // assert the output
    exp := `Hello from Garlic!
go.mod
go.sum
main.go`

    act := bb.String()
    act = strings.TrimSpace(act)

    r.Equal(exp, act)
}
```

Figure 1.52: cmd/bostongo/cli/garlic_test.go

Running the Tests

```
$ go test -v -run Garlic_Works

=== RUN   Test_Garlic_Works
=== PAUSE Test_Garlic_Works
=== CONT  Test_Garlic_Works
created garlic at /tmp/411396325/garlic/bostongo/main.go
go: finding module for package github.com/markbates/bostongo/cmd/bostongo/cli
go: found github.com/markbates/bostongo/cmd/bostongo/cli in github.com/markbates/bostongo
--- PASS: Test_Garlic_Works (0.94s)
PASS
ok      github.com/markbates/bostongo/cmd/bostongo/cli  1.197s

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.53: Running the `garlic` tests.

Final Folder Structure

```
$ tree -I testdata -I assets -I *.md

.
|-- LICENSE.txt
|-- Makefile
|-- cmd
|   |-- bostongo
|   |   |-- cli
|   |   |   |-- app.go
|   |   |   |-- app_test.go
|   |   |   |-- commands.go
|   |   |   |-- garlic.go
|   |   |   |-- garlic_test.go
|   |   |   `-- ifaces.go
|   |   `-- main.go
|   |-- server
|   |   |-- cli
|   |   |   |-- app.go
|   |   |   |-- app_test.go
|   |   |   |-- env.go
|   |   |   `-- env_test.go
|   |   `-- main.go
|   `-- walker
|       |-- cli
|       |   |-- app.go
|       |   `-- app_test.go
|       `-- main.go
|-- go.mod
|-- go.sum
|-- walker.go
|-- walker_test.go
`-- web
    |-- app.go
    |-- app_test.go
    `-- template.html

9 directories, 24 files
```

Figure 1.54: Final folder structure.

Summing Up

Avoid Globals

I/O

```
$ go doc github.com/markbates/iox.IO

package iox // import "github.com/markbates/iox"

type IO struct {
        In  io.Reader `json:"-"` // standard input
        Out io.Writer `json:"-"` // standard output
        Err io.Writer `json:"-"` // standard error
}
    IO represents the standard input, output, and error stream.

func Discard() IO
func (oi IO) Stderr() io.Writer
func (oi IO) Stdin() io.Reader
func (oi IO) Stdout() io.Writer

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.55: `iox.IO`

```go
type SettableIO interface {
    SetIO(io iox.IO)
}
```

Figure 1.56: cmd/bostongo/cli/ifaces.go

Environment Variables

```go
type Env struct {
    data map[string]string
    mu   sync.RWMutex
}

func (e *Env) Getenv(key string) string
func (e *Env) Setenv(key string, value string)
```

Figure 1.57: cmd/server/cli/env.go

The File System

```
$ go doc io/fs.FS

package fs // import "io/fs"

type FS interface {
        // Open opens the named file.
        //
        // When Open returns an error, it should be of type *PathError
        // with the Op field set to "open", the Path field set to name,
        // and the Err field describing the problem.
        //
        // Open should reject attempts to open names that do not satisfy
        // ValidPath(name), returning a *PathError with Err set to
        // ErrInvalid or ErrNotExist.
        Open(name string) (File, error)
}
    An FS provides access to a hierarchical file system.

    The FS interface is the minimum implementation required of the file system.
    A file system may implement additional interfaces, such as ReadFileFS,
    to provide additional or optimized functionality.

func Sub(fsys FS, dir string) (FS, error)

--------------------------------------------------------------------------------
Go Version: go1.21.0
```

Figure 1.58: `fs.FS`

```go
type SettableFS interface {
    SetFS(fs fs.FS)
}
```

Figure 1.59: `garlic.SettableFS`

Current Working Directory and Arguments

```go
type Commander interface {
    Main(ctx context.Context, pwd string, args []string) error
}
```

Figure 1.60: cmd/bostongo/cli/ifaces.go

Escape the `main` Package

```go
func main() {
    args := os.Args[1:]

    pwd, err := os.Getwd()
    if err != nil {
        log.Fatal(err)
    }

    ctx := context.Background()
    ctx, cancel := signal.NotifyContext(ctx, os.Interrupt)
    defer cancel()

    app := cli.App{}
    err = app.Main(ctx, pwd, args)

    if err != nil {
        log.Fatal(err)
    }
}
```

Figure 1.61: cmd/walker/main.go

Consider the Garlic Pattern

```go
func main() {
    args := os.Args[1:]

    pwd, err := os.Getwd()
    if err != nil {
        log.Fatal(err)
    }

    ctx, cancel := signal.NotifyContext(context.Background(), os.Interrupt)
    defer cancel()

    app := &cli.App{}

    clove := &garlic.Garlic{
        Name: app.PluginName(),
        Cmd:  app,
        FS:   os.DirFS(pwd),
    }

    err = clove.Main(ctx, pwd, args)
    if err != nil {
        log.Fatal(err)
    }
}
```

Figure 1.62: cmd/bostongo/main.go

Go Fundamentals

Gopher Guides