Go CLI Project Structure

Command-line application with Cobra, subcommands, and proper distribution setup.

#go #golang #cli #command-line #cobra

PNG PDF Copy Prompt

Project Directory

```
mycli/
    main.go  Entry point, ca…
    go.mod
    go.sum
    Makefile  Build, install,…
    LICENSE
    README.md
    .gitignore
    .goreleaser.yaml  Cross-platform …
  cmd/  CLI commands
    root.go  Root command an…
    init.go  mycli init
    run.go  mycli run
    config.go  mycli config
    version.go  mycli version
  internal/  Private impleme…
    config/
      config.go  Viper config lo…
    runner/  Core business l…
      runner.go
    output/
      printer.go  Table, JSON out…
  pkg/  Public utilities
    version/
      version.go  Build info via …
```

Why This Structure?

Cobra is the standard for Go CLIs (used by kubectl, hugo, gh). Each command lives in its own file in `cmd/`. The root command sets up global flags and config. Business logic stays in `internal/` to keep commands thin.

Key Directories

`cmd/` - One file per command (init.go, run.go)

`cmd/root.go` - Root command, global flags, Viper setup

`internal/` - Business logic, not command parsing

`.goreleaser.yaml` - Automated cross-platform builds

Cobra Root Command

```
// cmd/root.go
var rootCmd = &cobra.Command{
    Use:   "mycli",
    Short: "A brief description of your CLI",
}

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

func init() {
    cobra.OnInitialize(initConfig)
    rootCmd.PersistentFlags().StringVar(&cfgFile, "config"
}
```

Getting Started

1. `go mod init mycli`
2. `go get github.com/spf13/cobra`
3. `cobra-cli init`
4. `cobra-cli add run`
5. `go build -o mycli .`

When To Use This

- Building distributable CLI tools
- Commands with subcommands (like git, docker)
- Need --help generation and shell completion
- Config file support (YAML, TOML, JSON)
- Cross-platform distribution

Trade-offs

`Cobra overhead` - Larger binary than stdlib flag package

`Viper complexity` - Config merging can be confusing

`Code generation` - cobra-cli generates boilerplate you may not need

Naming Conventions

`Commands` - One file per command in cmd/ (run.go, init.go)

`Variables` - runCmd, initCmd for command vars

`Flags` - Descriptive names (--output, --verbose)

Testing Strategy

`Unit tests` - Test internal/ packages directly

`Command tests` - Execute commands and check output

`Golden files` - Compare output against expected files