

Professional Multi-file Go Project: File Integrity Checker (FIC)

Project Structure

```
fic/
├─ main.go
├─ hash.go
├─ verify.go
├─ utils.go
└─ go.mod
```

1 main.go – Entry Point & CLI

Purpose: - Program starts here (`package main` and `func main()`)

- Parses command-line arguments
- Calls functions in `hash.go` or `verify.go`
- Handles program flow (generate vs verify)

Code Responsibilities: - Read `os.Args` for user input

- Check if arguments are sufficient (`len(os.Args)`)
- Decide which command to run (`if / switch`)
- Call `computeSHA256()` from `hash.go` or `verifyFile()` from `verify.go`
- Print results or error messages

Prerequisites to Learn: - `package main` and `func main()`

- Command-line arguments (`os.Args`)
- Conditional statements (`if` , `switch`)
- Printing output (`fmt.Println` , `fmt.Printf`)
- Calling functions from other files in the same package

2 hash.go – Hashing Functions

Purpose: - Encapsulate all hash-related logic

- Compute SHA-256 of a file
- Can extend to other hash algorithms (MD5, SHA-1)

Code Responsibilities: - `computeSHA256(path string) (string, error)`

- Open file (`os.Open()`)

- Stream file to hasher (`io.Copy()`)
- Return hex-encoded hash
- Optional: hash strings or byte slices

Prerequisites to Learn: - Functions in Go (`func`)

- Returning multiple values (`return value, error`)
 - Error handling (`if err != nil`)
 - Opening files (`os.Open`) & `defer file.Close()`
 - `crypto/sha256` & `encoding/hex`
 - Streaming content with `io.Copy`
-

3 verify.go – Verification Functions

Purpose: - Encapsulate verification logic

- Compare computed hash to known hash
- Return success/failure result

Code Responsibilities: - `verifyFile(path string, knownHash string) (bool, error)`

- Calls `computeSHA256()` from `hash.go`
- Compares `computedHash == knownHash`
- Returns `true` if match, `false` if not
- Optional: batch verification, read hashes from file

Prerequisites to Learn: - Calling functions from other files

- String comparison (`==`)
 - Returning multiple values
 - Error handling and reporting
-

4 utils.go – Helper Functions

Purpose: - Optional, for reusable helpers

- Examples: `printUsage()`, logging, error formatting

Code Responsibilities: - Small generic helper functions

- Called from `main.go` or other files

Prerequisites to Learn: - Functions and scope in Go

- Exported vs unexported functions (`FuncName` vs `funcName`)
 - Reusing code to avoid duplication
-

5 go.mod – Module File

- Purpose:** - Defines your Go module
- Lets you import packages correctly
 - Needed for professional Go projects

How to Create:

```
go mod init fic
```

- Prerequisites to Learn:** - Go modules basics
- Package importing rules
 - Dependency management

Summary of Responsibilities

File	Purpose	Key Functions / Concepts
main.go	CLI, program entry, command parsing	os.Args, fmt, if/switch, calling hash/verify functions
hash.go	Compute SHA-256 (or other) hashes	os.Open, defer, io.Copy, crypto/sha256, encoding/hex, error handling
verify.go	Compare computed hash with known hash	String comparison, calling hash.go functions, return bool/error
utils.go	Helpers & reusable code	printUsage, logging, error formatting, reusable functions
go.mod	Go module file	<code>go mod init</code>

- Key Takeaways:** - Each file has a clear responsibility
- Learn the Go concepts for each file before coding
 - Keep project modular for maintainability, testability, and expansion