
depth

go report A+ go report A+ go report A+ coverage 92%

`depth` is tool to retrieve and visualize Go source code dependency trees.

Install

Download the appropriate binary for your platform from the Releases page, or:

```
1 go get github.com/KyleBanks/depth/cmd/depth
```

Usage

`depth` can be used as a standalone command-line application, or as a package within your own project.

Command-Line

Simply execute `depth` with one or more package names to visualize. You can use the fully qualified import path of the package, like so:

```
1 $ depth github.com/KyleBanks/depth/cmd/depth
2 github.com/KyleBanks/depth/cmd/depth├
3   encoding/json├
4   flag├
5   fmt├
6   io├
7   log├
8   os├
9   strings├
10  github.com/KyleBanks/depth├
11    fmt├
12    go/build├
13    path├
14    sort├
15    strings
16 12 dependencies (11 internal, 1 external, 0 testing).
```

Or you can use a relative path, for example:

```
1 $ depth .
2 $ depth ./cmd/depth
3 $ depth ../
```

You can also use `depth` on the Go standard library:

```
1 $ depth strings
2 strings└
3   errors└
4   internal/bytealg└
5   io└
6   sync└
7   unicode└
8   unicode/utf8└
9   unsafe
10 7 dependencies (7 internal, 0 external, 0 testing).
```

Visualizing multiple packages at a time is supported by simply naming the packages you'd like to visualize:

```
1 $ depth strings github.com/KyleBanks/depth
2 strings└
3   errors└
4   internal/bytealg└
5   io└
6   sync└
7   unicode└
8   unicode/utf8└
9   unsafe
10 7 dependencies (7 internal, 0 external, 0 testing).
11 github.com/KyleBanks/depth└
12   bytes└
13   errors└
14   go/build└
15   os└
16   path└
17   sort└
18   strings
19 7 dependencies (7 internal, 0 external, 0 testing).
```

-internal By default, `depth` only resolves the top level of dependencies for standard library packages, however you can use the `-internal` flag to visualize all internal dependencies:

```
1 $ depth -internal strings
2 strings└
```

```

3  errors|L
4  internal/reflectlite||-
5  internal/unsafeheader||L
6  unsafe||-
7  runtime||-
8  internal/abi|||L
9  unsafe||-
10 internal/bytealg|||
11 internal/cpu|||L
12 unsafe||-
13 internal/cpu||-
14 internal/goexperiment||-
15 runtime/internal/atomic|||L
16 unsafe||-
17 runtime/internal/math|||L
18 runtime/internal/sys||-
19 runtime/internal/sys||L
20 unsafe|L
21 unsafe|-
22 internal/bytealg|-
23 io||-
24 errors|L
25 sync||-
26 internal/race||L
27 unsafe||-
28 runtime||-
29 sync/atomic||L
30 unsafe|L
31 unsafe|-
32 sync|-
33 unicode|-
34 unicode/utf8L
35 unsafe
36 18 dependencies (18 internal, 0 external, 0 testing).

```

-max The **-max** flag limits the dependency tree to the maximum depth provided. For example, if you supply **-max 1** on the **depth** package, your output would look like so:

```

1 $ depth -max 1 github.com/KyleBanks/depth/cmd/depth
2 github.com/KyleBanks/depth/cmd/depth|-
3 encoding/json|-
4 flag|-
5 fmt|-

```

```
6   io|
7   log|
8   os|
9   stringsL
10  github.com/KyleBanks/depth
11  7 dependencies (6 internal, 1 external, 0 testing).
```

The `-max` flag is particularly useful in conjunction with the `-internal` flag which can lead to very deep dependency trees.

-test By default, `depth` ignores dependencies that are only required for testing. However, you can view test dependencies using the `-test` flag:

```
1  $ depth -test strings
2  strings|
3    bytes|
4    errors|
5    fmt|
6    internal/bytealg|
7    internal/testenv|
8    io|
9    math/rand|
10   reflect|
11   strconv|
12   sync|
13   testing|
14   unicode|
15   unicode/utf8L
16   unsafe
17  14 dependencies (14 internal, 0 external, 7 testing).
```

-explain target-package The `-explain` flag instructs `depth` to print import chains in which the `target-package` is found:

```
1  $ depth -explain strings github.com/KyleBanks/depth/cmd/depth
2  github.com/KyleBanks/depth/cmd/depth -> strings
3  github.com/KyleBanks/depth/cmd/depth -> github.com/KyleBanks/depth ->
    strings
```

-json The `-json` flag instructs `depth` to output dependencies in JSON format:

```
1  $ depth -json github.com/KyleBanks/depth/cmd/depth
2  {
```

```
3  "name": "github.com/KyleBanks/depth/cmd/depth",
4  "deps": [
5    {
6      "name": "encoding/json",
7      "internal": true,
8      "deps": null
9    },
10   ...
11   {
12     "name": "github.com/KyleBanks/depth",
13     "internal": false,
14     "deps": [
15       {
16         "name": "go/build",
17         "internal": true,
18         "deps": null
19       },
20       ...
21     ]
22   }
23 ]
24 }
```

Integrating With Your Project

The `depth` package can easily be used to retrieve the dependency tree for a particular package in your own project. For example, here's how you would retrieve the dependency tree for the `strings` package:

```
1  import "github.com/KyleBanks/depth"
2
3  var t depth.Tree
4  err := t.Resolve("strings")
5  if err != nil {
6      log.Fatal(err)
7  }
8
9  // Output: "'strings' has 4 dependencies."
10 log.Printf("'%v' has %v dependencies.", t.Root.Name, len(t.Root.Deps))
```

For additional customization, simply set the appropriate flags on the `Tree` before resolving:

```
1  import "github.com/KyleBanks/depth"
2
3  t := depth.Tree {
4      ResolveInternal: true,
5      ResolveTest: true,
6      MaxDepth: 10,
```

```
7 }  
8  
9  
10 err := t.Resolve("strings")
```

Author

`depth` was developed by Kyle Banks.

License

`depth` is available under the MIT license.