

# treeprint

Package `treeprint` provides a simple ASCII tree composing tool.

If you are familiar with the [tree](#) utility that is a recursive directory listing command that produces a depth indented listing of files, then you have the idea of what it would look like.

On my system the command yields the following

```
$ tree
.
├── LICENSE
├── README.md
├── treeprint.go
└── treeprint test.go
```

```
0 directories, 4 files
```

and I'd like to have the same format for my Go data structures when I print them.

## Installation

```
$ go get github.com/xlab/treeprint
```

## Concept of work

The general idea is that you initialise a new tree with `treeprint.New()` and then add nodes and branches into it. Use `AddNode()` when you want add a node on the same level as the target or use `AddBranch()` when you want to go a level deeper. So `tree.AddBranch().AddNode().AddNode()` would create a new level with two distinct nodes on it. So `tree.AddNode().AddNode()` is a flat thing and `tree.AddBranch().AddBranch().AddBranch()` is a high thing. Use `String()` or `Bytes()` on a branch to render a subtree, or use it on the root to print the whole tree.

The utility will yield Unicode-friendly trees. The output is predictable and there is no platform-dependent exceptions, so if you have issues with displaying the tree in the console, all platform-related transformations can be done after the tree has been rendered: [an example](#) for Asian locales.

## Use cases

When you want to render a complex data structure:

```
func main() {  
    tree := treeprint.New()  
  
    // create a new branch in the root
```

```

one := tree.AddBranch("one")

// add some nodes
one.AddNode("subnode1").AddNode("subnode2")

// create a new sub-branch
one.AddBranch("two").
    AddNode("subnode1").AddNode("subnode2"). // add some nodes
    AddBranch("three"). // add a new sub-branch
    AddNode("subnode1").AddNode("subnode2") // add some nodes too

// add one more node that should surround the inner branch
one.AddNode("subnode3")

// add a new node to the root
tree.AddNode("outernode")

fmt.Println(tree.String())
}

```

Will give you:

```

.
├─ one
│   ├── subnode1
│   ├── subnode2
│   └─ two
│       ├── subnode1
│       ├── subnode2
│       └─ three
│           ├── subnode1
│           └─ subnode2
└─ subnode3
└─ outernode

```

Another case, when you have to make a tree where any leaf may have some meta-data (as `tree` is capable of it):

```

func main {
    tree := treeprint.New()

    tree.AddNode("Dockerfile")
    tree.AddNode("Makefile")
    tree.AddNode("aws.sh")
    tree.AddMetaBranch(" 204", "bin").
        AddNode("dbmaker").AddNode("someserver").AddNode("testtool")
    tree.AddMetaBranch(" 374", "deploy").
        AddNode("Makefile").AddNode("bootstrap.sh")
    tree.AddMetaNode("122K", "testtool.a")

    fmt.Println(tree.String())
}

```

Output:

```
.
├─ Dockerfile
├─ Makefile
├─ aws.sh
├─ [ 204]  bin
│   └─ dbmaker
│   └─ someserver
│   └─ testtool
├─ [ 374]  deploy
│   └─ Makefile
│   └─ bootstrap.sh
└─ [122K]  testtool.a
```

Yay! So it works.

## License

MIT