# Golang tools

Go offers a powerful set of command line tools which makes the life of a developer easier. The syntax for using command line tools is **go commandname arguments**

We will discuss about the most commonly used go tools.

# get

get is used to download and install packages and dependencies. Lets assume you want to use the **logrus** logging package from github present at location https://github.com/sirupsen/logrus in your code.

The command **go get github.com/sirupsen/logrus** will download the package and install it for you. After running this command, you can see that the source code is downloaded inside the local go workspace directory under the path **github.com/sirupsen/logrus**.

Now *import "github.com/sirupsen/logrus"* can be used in the code to import the package and use it.

# fmt

fmt is used to format go source code. It uses tabs for indentation and blanks for alignment. Its input can either be a source file or a directory. If it's a directory, it formats all the go files in it.

Consider the following go program

```
package main

import "fmt"

func main() {
fmt.Println("Hello world")
}
```

In the above program indentation is wrong for the line of code *fmt.Println("Hello world").* If you run **go fmt /pathtofile/filename.go,** the go source file will be re-formatted as shown below and saved.

```
package main

import "fmt"

func main() {
   fmt.Println("Hello world")
}
```

# doc

doc is used to print the documentation associated with the package, constant, ... depending on what is passed as argument to the command.

For example **go doc fmt** will print the documentation of the fmt package.

# version

version is used to print the go version. Running **go version** will print something similar to **go version go1.8 darwin/amd64** depending on your installed version.

# vet

vet analyses the go source code and reports possible suspicious code. Everything reported by vet is not a genuine problem but it has the capability to catch errors that are not reported by the compiler.

Consider the following program,

```
package main

import "fmt"

func main() {
   var i int = 10
   fmt.Printf("Hello world %f", i)
}
```

The format specifier in the Printf should be %d for int whereas its specified as %f in the printf. This will pass compilation without any errors. However if you run **go vet /pathtofile/filename.go**, you will get the following output.

**arg i for printf verb %f of wrong type: int**

**exit status 1**

# env

**go env** will print the go environment information. Running **go env** will print something similar to the following output.

*GOARCH="amd64"*

*GOBIN=""*

*GOEXE=""*

*GOHOSTARCH="amd64"*

*GOHOSTOS="darwin"*

*GOOS="darwin"*

*GOPATH="/Users/naveen/go"*

*GORACE=""*

*GOROOT="/usr/local/go"*

If go env is given a environment variable as argument, it will print the value for that variable. For instance **go env GOPATH** will print **/Users/naveen/go**

# list

**list** is used to display the import path of the current directory. For instance if you are present inside the built in zip package (In my Mac its located at **/usr/local/go/src/archive/zip/)** and then execute **go list** inside that directory, **archive/zip** will be displayed.

This is not of much use, but the list command comes with lots of useful arguments. One such argument is **...** which lists all the packages. The command **go list ...** will list all packages. Running this command in my Mac presents me a list of the form,

*archive/tar*

*archive/zip*

*bufio*

*…*

*golang.org/x/tools/refactor/satisfy*

Your list will be different from mine :).