How To Install Go and Set Up a Local Programming Environment on Ubuntu 18.04

Introduction

Go is a programming language that was born out of frustration at Google. Developers continually had to pick a language that executed efficiently but took a long time to compile, or to pick a language that was easy to program but ran inefficiently in production. Go was designed to have all three available at the same time: fast compilation, ease of programming, and efficient execution in production.

While Go is a versatile programming language that can be used for many different programming projects, it's particularly well suited for networking/distributed systems programs, and has earned a reputation as "the language of the cloud". It focuses on helping the modern programmer do more with a strong set of tooling, removing debates over formatting by making the format part of the language specification, as well as making deployment easy by compiling to a single binary. Go is easy to learn, with a very small set of keywords, which makes it a great choice for beginners and experienced developers alike.

This tutorial will guide you through installing and configuring a programming workspace with Go via the command line. This tutorial will explicitly cover the installation procedure for Ubuntu 18.04, but the general principles can apply to other Debian Linux distributions.

Prerequisites

You will need a setupcomputer or virtual machine with Ubuntu 18.04 installed, as well as have administrative access to that machine and an internet connection. You can download this operating system via the Ubuntu 18.04 releases page.

Step 1 — Setting Up Go

In this step, you'll install Go by downloading the current release from the official Go downloads page.

To do this, you'll want to find the URL for the current binary release tarball. You will also want to note the SHA256 hash listed next to it, as you'll use this hash to verify the downloaded file.

You'll be completing the installation and setup on the command line, which is a non-graphical way to interact with your computer. That is, instead of clicking on buttons, you'll be typing in text and receiving feedback from your computer through text as well.

The command line, also known as a *shell* or *terminal*, can help you modify and automate many of the tasks you do on a computer every day, and is an essential tool for software developers. There are many terminal commands to learn that can enable you to do more powerful things. For more information about the command line, check out the Introduction to the Linux Terminal tutorial.

On Ubuntu 18.04, you can find the Terminal application by clicking on the Ubuntu icon in the upper-left hand corner of your screen and typing `terminal` into the search bar. Click on the Terminal application icon to open it. Alternatively, you can hit the `CTRL`, `ALT`, and `T` keys on your keyboard at the same time to open the Terminal application automatically.

Once the terminal is open, you will manually install the Go binaries. While you could use a package manager, such as `apt-get`, walking through the manual installation steps will help you understand any configuration changes to your system that are needed to have a valid Go workspace.

Before downloading Go, make sure that you are in the home (`~`) directory:

```
cd ~
```

Copy

Use `curl` to retrieve the tarball URL that you copied from the official Go downloads page:

```
curl -LO https://dl.google.com/go/go1.12.1.linux-amd64.tar.gz
```

Copy

Next, use `sha256sum` to verify the tarball:

```
sha256sum go1.12.1.linux-amd64.tar.gz
```

Copy

The hash that is displayed from running the above command should match the hash that was on the downloads page. If it does not, then this is not a valid file and you should download the file again.

```
Output
```

```
2a3fdabf665496a0db5f41ec6af7a9b15a49fbe71a85a50ca38b1f13a103aeec  go1.12.1.linux-amd64.tar.gz
```

Next, extract the downloaded archive and install it to the desired location on the system. It's considered best practice to keep it under `/usr/local`:

```
sudo tar -xvf go1.12.1.linux-amd64.tar.gz -C /usr/local
```

Copy

You will now have a directory called `go` in the `/usr/local` directory.

Note: Although `/usr/local/go` is the officially-recommended location, some users may prefer or require different paths.

In this step, you downloaded and installed Go on your Ubuntu 18.04 machine. In the next step you will configure your Go workspace.

Step 2 — Creating Your Go Workspace

You can create your programming workspace now that Go is installed. The Go workspace will contain two directories at its root:

- `src`: The directory that contains Go source files. A source file is a file that you write using the Go programming language. Source files are used by the Go compiler to create an executable binary file.
- `bin`: The directory that contains executables built and installed by the Go tools. Executables are binary files that run on your system and execute tasks. These are typically the programs compiled by your source code or other downloaded Go source code.

The `src` subdirectory may contain multiple version control repositories (such as Git, Mercurial, and Bazaar). This allows for a canonical import of code in your project. *Canonical* imports are imports that reference a fully qualified package, such as `github.com/digitalocean/godo`.

You will see directories like `github.com`, `golang.org`, or others when your program imports third party libraries. If you are using a code repository like `github.com`, you will also put your projects and source files under that directory. We will explore this concept later in this step.

Here is what a typical workspace may look like:

```
.
├── bin
│   ├── buffalo                                    # command executable
│   ├── dlv                                        # command executable
│   └── packr                                      # command executable
└── src
    └── github.com
        └── digitalocean
            └── godo
                ├── .git                           # Git repository metadata
                ├── account.go                     # package source
```

```
            ├── account_test.go                 # test source
            ├── ...
            ├── timestamp.go
            ├── timestamp_test.go
            └── util
                ├── droplet.go
                └── droplet_test.go
```

The default directory for the Go workspace as of 1.8 is your user's home directory with a `go` subdirectory, or `$HOME/go`. If you are using an earlier version of Go than 1.8, it is still considered best practice to use the `$HOME/go` location for your workspace.

Issue the following command to create the directory structure for your Go workspace:

```
mkdir -p $HOME/go/{bin,src}
```

Copy

The `-p` option tells `mkdir` to create all `parents` in the directory, even if they don't currently exist. Using `{bin,src}` creates a set of arguments to `mkdir` and tells it to create both the `bin` directory and the `src` directory.

This will ensure the following directory structure is now in place:

```
└── $HOME
```

```
└── go
├── bin
└── src
```

Prior to Go 1.8, it was required to set a local environment variable called `$GOPATH`. `$GOPATH` told the compiler where to find imported third party source code, as well as any local source code you had written. While it is no longer explicitly required, it is still considered a good practice as many third party tools still depend on this variable being set.

You can set your `$GOPATH` by adding the global variables to your `~/.profile`. You may want to add this into `.zshrc` or `.bashrc` file as per your shell configuration.

First, open `~/.profile` with `nano` or your preferred text editor:

```
nano ~/.profile
```

Copy

Set your `$GOPATH` by adding the following to the file:

~/.profile

```
export GOPATH=$HOME/go
```

Copy

When Go compiles and installs tools, it will put them in the `$GOPATH/bin` directory. For convenience, it's common to add the workspace's `/bin` subdirectory to your `PATH` in your `~/.profile`:

~/.profile

```
export PATH=$PATH:$GOPATH/bin
```

Copy

This will allow you to run any programs you compile or download via the Go tools anywhere on your system.

Finally, you need to add the `go` binary to your `PATH`. You can do this by adding `/usr/local/go/bin` to the end of the line:

~/.profile

```
export PATH=$PATH:$GOPATH/bin:/usr/local/go/bin
```

Copy

Adding `/usr/local/go/bin` to your `$PATH` makes all of the Go tools available anywhere on your system.

To update your shell, issue the following command to load the global variables:

```
. ~/.profile
```

Copy

You can verify your `$PATH` is updated by using the `echo` command and inspecting the output:

```
echo $PATH
```

Copy

You will see your `$GOPATH/bin` which will show up in your home directory. If you are logged in as `root`, you would see `/root/go/bin` in the path.

`Output`

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap
root/go/bin:/usr/local/go/bin
```

You will also see the path to the Go tools for `/usr/local/go/bin`:

`Output`

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap
root/go/bin:/usr/local/go/bin
```

Verify the installation by checking the current version of Go:

```
go version
```

Copy

And we should receive output like this:

`Output`

`go version go1.12.1 linux/amd64`

Now that you have the root of the workspace created and your `$GOPATH` environment variable set, you can create your future projects with the following directory structure. This example assumes you are using `github.com` as your repository:

`$GOPATH/src/github.com/username/project`

So as an example, if you were working on the `https://github.com/digitalocean/godo` project, it would be stored in the following directory:

`$GOPATH/src/github.com/digitalocean/godo`

This project structure will make projects available with the `go get` tool. It will also help readability later. You can verify this by using the `go get` command and fetch the `godo` library:

```
go get github.com/digitalocean/godo
```

Copy

This will download the contents of the `godo` library and create the `$GOPATH/src/github.com/digitalocean/godo` directory on your machine.

You can check to see if it successfully downloaded the `godo` package by listing the directory:

```
ll $GOPATH/src/github.com/digitalocean/godo
```

Copy

You should see output similar to this:

```
Output

drwxr-xr-x 4 root root  4096 Apr  5 00:43 ./

drwxr-xr-x 3 root root  4096 Apr  5 00:43 ../

drwxr-xr-x 8 root root  4096 Apr  5 00:43 .git/

-rwxr-xr-x 1 root root   8 Apr  5 00:43 .gitignore*

-rw-r--r-- 1 root root   61 Apr  5 00:43 .travis.yml

-rw-r--r-- 1 root root  2808 Apr  5 00:43 CHANGELOG.md

-rw-r--r-- 1 root root  1851 Apr  5 00:43 CONTRIBUTING.md

.

.

.

-rw-r--r-- 1 root root  4893 Apr  5 00:43 vpcs.go

-rw-r--r-- 1 root root  4091 Apr  5 00:43 vpcs_test.go
```

In this step, you created a Go workspace and configured the necessary environment variables. In the next step you will test the workspace with some code.

Step 3 — Creating a Simple Program

Now that you have the Go workspace set up, create a “Hello, World!” program. This will make sure that the workspace is configured properly, and also gives you the opportunity to become more familiar with Go. Because we are creating a single Go source file, and not an actual project, we don’t need to be in our workspace to do this.

From your home directory, open up a command-line text editor, such as `nano`, and create a new file:

```
nano hello.go
```

Copy

Write your program in the new file:

```
package main


import "fmt"


func main() {
    fmt.Println("Hello, World!")
}
```

Copy

This code will use the `fmt` package and call the `Println` function with `Hello, World!` as the argument. This will cause the phrase `Hello, World!` to print out to the terminal when the program is run.

Exit `nano` by pressing the `CTRL` and `X` keys. When prompted to save the file, press `Y` and then `ENTER`.

Once you exit out of `nano` and return to your shell, run the program:

```
go run hello.go
```

The `hello.go` program will cause the terminal to produce the following output:

```
Output
```

```
Hello, World!
```

In this step, you used a basic program to verify that your Go workspace is properly configured.

Conclusion

Congratulations! At this point you have a Go programming workspace set up on your Ubuntu machine and can begin a coding project!