Red Hat Developer Tools 2018.4

Using Go Toolset

Installing and using Go Toolset 1.10

Last Updated: 2018-11-29

Red Hat Developer Tools 2018.4 Using Go Toolset

Installing and using Go Toolset 1.10

Robin Owen
kowen@redhat.com

Legal Notice

Abstract

Go Toolset is a Red Hat offering for developers on the Red Hat Enterprise Linux platform. The Using Go Toolset provides an overview of this product, explains how to invoke and use the Go Toolset versions of the tools, and links to resources with more in-depth information.

Table of Contents

CHAPTER 1. GO TOOLSET

1.1. ABOUT GO TOOLSET

Go Toolset is a Red Hat offering for developers on the Red Hat Enterprise Linux platform. It provides the Go programming language tools and libraries. Go is alternatively known as **golang**.

Go Toolset is distributed as a part of Red Hat Developer Tools for Red Hat Enterprise Linux 7.

The following components are available as a part of Go Toolset:

Table 1.1. Go Toolset 1.10.3 Components

Name	Version	Description
golang	1.10.3	A Go compiler.

1.2. COMPATIBILITY

Go Toolset is available for Red Hat Enterprise Linux 7 on the following architectures:

- The 64-bit Intel and AMD architectures
- The 64-bit ARM architecture
- The IBM Power Systems architecture
- The little-endian variant of IBM Power Systems architecture
- The IBM Z Systems architecture

1.3. GETTING ACCESS TO GO TOOLSET

Go Toolset is an offering that is distributed as a part of the Red Hat Developer Tools content set, which is available to customers with deployments of Red Hat Enterprise Linux 7. In order to install Go Toolset, enable the Red Hat Developer Tools and Red Hat Software Collections repositories by using the Red Hat Subscription Management and add the Red Hat Developer Tools key to your system.

1. Enable the **`rhel-7-variant-devtools-rpms`** repository:

   ```
   # subscription-manager repos --enable rhel-7-variant-devtools-rpms
   ```

   Replace *variant* with the Red Hat Enterprise Linux system variant (**`server`** or **`workstation`**).

   NOTE

   We recommend developers to use Red Hat Enterprise Linux Server for access to the widest range of development tools.

2. Enable the **`rhel-variant-rhscl-8-rpms`** repository:

   ```
   # subscription-manager repos --enable rhel-variant-rhscl-8-rpms
   ```

Replace *variant* with the Red Hat Enterprise Linux system variant (**`server`** or **`workstation`**).

3. Add the Red Hat Developer Tools key to your system:

```
# cd /etc/pki/rpm-gpg
# wget -O RPM-GPG-KEY-redhat-devel
https://www.redhat.com/security/data/a5787476.txt
# rpm --import RPM-GPG-KEY-redhat-devel
```

Once the subscription is attached to the system, and repositories enabled you can install Red Hat Go Toolset as described in Section 1.4, “Installing Go Toolset”.

Additional Resources

- For more information on how to register your system using Red Hat Subscription Management and associate it with subscriptions, see the Red Hat Subscription Management collection of guides.
- For detailed instructions on subscription to Red Hat Software Collections, see the *Red Hat Developer Toolset User Guide*, Section 1.4. Getting Access to Red Hat Developer Toolset.

1.4. INSTALLING GO TOOLSET

Go Toolset is distributed as a collection of RPM packages that can be installed, updated, uninstalled, and inspected by using the standard package management tools that are included in Red Hat Enterprise Linux. Note that a valid subscription that provides access to the Red Hat Developer Tools content set is required in order to install Go Toolset on your system. For detailed instructions on how to associate your system with an appropriate subscription and get access to Go Toolset, see Section 1.3, “Getting Access to Go Toolset”.

IMPORTANT

Before installing Go Toolset, install all available Red Hat Enterprise Linux updates.

1. To install all components that are included in Go Toolset, install the **go-toolset-1.10** package:

```
# yum install go-toolset-1.10
```

This installs all development and debugging tools, and other dependent packages to the system.

2. Create the Go language workspace directory and environment variable:

```
$ mkdir -p workspace_dir
$ echo 'export GOPATH=workspace_dir' >> $HOME/.bashrc
$ source $HOME/.bashrc
```

Select an appropriate value for the ***`workspace_dir`*** directory. A common choice is **`$HOME/go`**.

If the **`GOPATH`** variable is not set, the **go** compiler uses the **`~/go`** directory.

Additional Resources

- Workspaces — Description of the Go language workspace organization. Official documentation for the Go programming language.

1.5. ADDITIONAL RESOURCES

A detailed description of the Go Toolset and all its features is beyond the scope of this book. For more information, see the resources listed below.

Online Documentation

- The Go programming language Documentation — Official documentation for the Go programming language, tools, and libraries.

CHAPTER 2. GO

go is a build tool and dependency manager for the Go programming language.

Go Toolset is distributed with **go 1.10.3**.

2.1. INSTALLING GO

In Go Toolset, **go** is provided by the **go-toolset-1.10-golang** package and is automatically installed with the **go-toolset-1.10** package. See Section 1.4, “Installing Go Toolset”.

2.2. WRITING GO 1.10.3 PROGRAMS

When creating a Go program, developers must follow the rules for Go workspace layout. The **`.go`** source files must be placed in subdirectory of **`$GOPATH/src`**.

Example 2.1. Creating a Go Program

Consider a program named **`hello`** consisting of a single source file named **`hello.go`**:

```
$ mkdir -p $GOPATH/src/hello
$ cd $GOPATH/src/hello
$ touch hello.go
```

Edit the file **`hello.go`** in your favorite text editor to add the following text:

```
package main

import (
    "fmt"
    "net/http"
)

func Welcome(w http.ResponseWriter, req *http.Request) {

    fmt.Fprintf(w, "<h1>Welcome to the Go toolset.</h1>")

}

func main() {

    fmt.Println("Hello.")
    fmt.Println("Starting http server.")
    // Register handler function
    http.HandleFunc("/welcome", Welcome)
    fmt.Println("Go to localhost:8080/welcome")
    fmt.Println("To terminate press CTRL+C.")
    // Start server
    http.ListenAndServe(":8080", nil)

}
```

Additional Resources

- Workspaces — Description of the Go language workspace organization. Official documentation for the Go programming language.

2.3. USING THE GO COMPILER

To build a Go program using the command line, change to the project directory and run the **go** compiler as follows:

```
$ scl enable go-toolset-1.10 'go build -o output_file go_main_package'
```

This creates a binary file named ***output_file*** in the current working directory. If the **-o** option is omitted, the compiler creates a file named after the *go_main_package*, ***go_main_package***.

If *go_main_package* is not a main package or if multiple projects or ***.go** files are specified, the resulting binaries are discarded. In that case, the **go build** command is used to verify that the supplied projects or files can be built.

Note that you can execute any command using the **scl** utility, causing it to be run with the Go Toolset binaries available. This allows you to run a shell session with Go Toolset **go** directly available:

```
$ scl enable go-toolset-1.10 'bash'
```

Example 2.2. Compiling a Go Program Using the Command Line

Assuming that you have successfully created the program **hello** as shown in Example 2.1, “Creating a Go Program”, compile the program:

```
$ scl enable go-toolset-1.10 'go build hello.go'
```

This creates a new binary file called **hello** in the current working directory.

2.4. RUNNING A GO PROGRAM

When **go** compiles a program, it creates an executable binary file. To run this program on the command line, change to the directory with the executable file and run the program:

```
$ ./file_name
```

Example 2.3. Running a Go Program on the Command Line

Assuming that you have successfully compiled the **hello** binary file as shown in Example 2.2, “Compiling a Go Program Using the Command Line”, run it by typing the following at a shell prompt:

```
$ ./hello
Hello.
Starting http server.
Go to localhost:8080/welcome
To terminate press CTRL+C.
```

2.5. INSTALLING GO PROJECTS

Installing a Go project means that its executable files and libraries are compiled, and copied to appropriate directories in the Go Workspace. The **go** tool can then use the executable files and libraries in further projects. Dependencies of the installed project are installed, too.

To install a Go project, run the **go** tool:

```
$ scl enable go-toolset-1.10 'go install go_project'
```

The **`install`** command accepts the same options as the **`build`** command.

2.6. DOWNLOADING GO PROJECTS

To download a 3rd party Go project from an online source and install it, run the **go** tool:

```
$ scl enable go-toolset-1.10 'go get 3rd_party_go_project'
```

For more details about the possible values of *3rd_party_go_project* option, run the following command:

```
$ scl enable go-toolset-1.10 'go help importpath'
```

2.7. ADDITIONAL RESOURCES

A detailed description of the **go** compiler and its features is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

- The Go compiler **`help`** command provides information on its usage. To show the help index:

  ```
  $ scl enable go-toolset-1.10 'go help'
  ```

- The Go compiler **doc** command shows documentation for packages. To show documentation for package *package_name*:

  ```
  $ scl enable go-toolset-1.10 'go doc package_name'
  ```

  To learn more about the **doc** command:

  ```
  $ scl enable go-toolset-1.10 'go help doc'
  ```

Online Documentation

- Command go — Official documentation of the **go** compiler.

See Also

- Chapter 1, *Go Toolset* — An overview of Go Toolset and more information on how to install it on your system.

CHAPTER 3. GOFMT

gofmt is a code formatting tool for the Go programming language, packaged together with the **go** compiler.

Go Toolset is distributed with **gofmt 1.10.3**.

3.1. INSTALLING GOFMT

In Go Toolset, **gofmt** is provided by the **go-toolset-1.10-golang** package and is automatically installed with the **go-toolset-1.10** package. See Section 1.4, “Installing Go Toolset”.

3.2. FORMATTING CODE

To format all code in the *code_path* path, run the **gofmt** tool as follows:

```
$ scl enable go-toolset-1.10 'gofmt -w code_path'
```

This command will directly change code in the ***code_path*** path. When ***code_path*** is a single file, the changes apply only to the file. When ***code_path*** is a directory, all **.go** files in the directory are processed.

When the *code_path* is omitted, **gofmt** reads standard input instead.

To print the formatted code to standard output instead of writing to the original file, omit the **-w** option.

It is possible to invoke **gofmt** through the **go** compiler with the **fmt** command. To achieve the same results as the command above, run:

```
$ scl enable go-toolset-1.10 'go fmt -l -w -s code_path'
```

3.3. PREVIEWING CHANGES TO CODE

To preview changes done by formatting code in a given path ***code_path***, run the **gofmt** tool with the **-d** option as follows:

```
$ scl enable go-toolset-1.10 'gofmt -d code_path'
```

The output in unified diff format is printed to standard output.

It is possible to combine both the **-d** and **-w** options.

3.4. SIMPLIFYING CODE

To simplify code in a given path ***code_path***, run the **gofmt** tool with the **-s** option as follows:

```
$ scl enable go-toolset-1.10 'gofmt -s code_path'
```

The code under ***code_path*** is simplified. Use the **-d** option to show the differences, and use the **-w** option to apply the changes to the code.

3.5. REFACTORING CODE

The **gofmt** tool can be used to refactor code by applying arbitrary substitutions. To refactor code in a given path ***code_path*** according to a rule *rewrite_rule*, run the **gofmt** tool with the **-r** option as follows:

```
$ scl enable go-toolset-1.10 'gofmt -r rewrite_rule code_path'
```

The code under ***code_path*** is refactored according to the rule *rewrite_rule*. Use the **-d** option to show the differences, and use the **-w** option to apply the changes to the code. The additional options must be placed after the rule *rewrite_rule*:

```
$ scl enable go-toolset-1.10 'gofmt -r rewrite_rule -d code_path'
```

Detailed description of the rewrite rules is beyond the scope of this book. For more information, see the resources listed in Section 3.6, “Additional Resources”.

3.6. ADDITIONAL RESOURCES

A detailed description of the **gofmt** tool and its features is beyond the scope of this book. For more information, see the resources listed below.

Online Documentation

- Command gofmt — Official documentation of the **gofmt** tool.

See Also

- Chapter 1, *Go Toolset* — An overview of Go Toolset and more information on how to install it on your system.

CHAPTER 4. GO RACE DETECTOR

Go Toolset includes the Go race detector. The race detector is a feature of the Go standard library. The race detector must be enabled at compile time and is used at runtime.

4.1. INSTALLING THE RACE DETECTOR

In Go Toolset, the race detector is provided by the **go-toolset-1.10-golang-race** package. To install the race detector:

```
# yum install go-toolset-1.10-golang-race
```

A variant of the Go standard library with the runtime race detection enabled is installed.

See Also

- Section 1.4, “Installing Go Toolset”.

4.2. USING THE RACE DETECTOR

To use the runtime race detector for a Go project, add the **`-race`** option to the **go** tool commands used when manipulating the project.

For a minimal approach to using the race detector, build the project with the **`-race`** option:

```
$ scl enable go-toolset-1.10 'go build -race -o output_file
go_main_package'
```

When you run the resulting executable, the race detector will print warnings to the standard output when a race is detected.

IMPORTANT

The race detector has a significant runtime resource overhead.

4.3. ADDITIONAL RESOURCES

A detailed description of the Go race detector and its features is beyond the scope of this book. For more information, see the resources listed below.

Online Documentation

- Data Race Detector — Official documentation of the Go race detector.

See Also

- Chapter 1, *Go Toolset* — An overview of Go Toolset and more information on how to install it on your system.

CHAPTER 5. CONTAINER IMAGE

The Go Toolset is available as a Docker-formatted container image which can be downloaded from Red Hat Container Registry.

5.1. IMAGE CONTENTS

The **devtools/go-toolset-1.10-rhel7** image provides content corresponding to the following packages:

Component	Version	Package
Go	1.10.3	go-toolset-1.10

5.2. ACCESS TO THE IMAGE

To pull the **devtools/go-toolset-1.10-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/devtools/go-toolset-1.10-rhel7
```

5.3. USAGE AS BUILDER IMAGE WITH SOURCE-TO-IMAGE

The Go Toolset docker container image is prepared for use as a Source-to-Image (S2I) builder image.

To do so, set the following build environment variables:

`IMPORT_URL`

Set this variable to an URL specifying the location of the code. The rules for the `go get` command option apply.

`INSTALL_URL`

Set this variable to an URL specifying the location of the package that will provide the application's main executable file when built. The rules for the `go install` command option apply.
This variable can be omitted if the main package location is identical with the location specified by the **`IMPORT_URL`** variable.

Example 5.1. Building a Go Application Image Using Source-to-Image

To build the **md2man** package from its GitHub repository:

```
$ s2i build -e IMPORT_URL='github.com/cpuguy83/go-md2man' -e
INSTALL_URL='github.com/cpuguy83/go-md2man'
git://github.com/cpuguy83/go-md2man
registry.access.redhat.com/devtools/go-toolset-8-rhel7 md2man-app
```

A locally available application image **md2man-app** is built from the repository on GitHub using the **go-toolset-8** Docker image.

To fully leverage the Go Toolset as a S2I builder image, build custom images based on it, with modified S2I assemble scripts and further modifications to accomodate the particular application being built.

A detailed description of the Go Toolset usage with Source-to-Image is beyond the scope of this book. For more information about Source-to-Image, refer to the *OpenShift Container Platform 3.7 Image Creation Guide*, Chapter 4. S2I Requirements and *Using Red Hat Software Collections Container Images*, Chapter 2. Using Source-to-Image (S2I).

5.4. ADDITIONAL RESOURCES

- Go Toolset 8 - entry in the Red Hat Container Catalog
- Using Red Hat Software Collections Container Images

CHAPTER 6. CHANGES IN GO TOOLSET IN RED HAT DEVELOPER TOOLS 2018.4

This chapter lists some notable changes in Go Toolset since its previous release.

6.1. GO

Go has been updated from version **1.10.2** to **1.10.3**. Notable changes include:

- Go programs built with Go Toolset are FIPS compliant.
  The cryptographic library available in Go Toolset has been changed to link against the OpenSSL library version 1.0.2. As a consequence, the Go cryptographic libraries are FIPS compliant. As a result, programs built with this version of Go Toolset are FIPS compliant, too. The Go **`crypto`** library will call into OpenSSL for FIPS compliance if the host system is also configured in FIPS mode.

  The optional compiler flag **`-tags no_openssl`** can be used to disable this and build with pure Go cryptography. For example, during a normal build using this toolchain, invoke the Go tool in the following way to disable FIPS compliance and dynamically linking against OpenSSL:

  ```
  $ go build -tags no_openssl
  ```