Go的官方仓库地址：

https://go.googlesource.com/go

Go的github的仓库地址：

https://github.com/golang/go

Go的官网地址：

https://golang.org

按照网上的教程，来安装Go时，报错，出现了：

ERROR: Cannot find /root/go1.4/bin/go.

Set $GOROOT\_BOOTSTRAP to a working Go tree >= Go 1.4.

查看了下 make.bash，里面是在 '$HOME/go1.4'，查找 'Go编译器'，确实并未找到。

网上查看了相关的资料＋官方文档，做个笔记记录下来：

参考文章：

https://github.com/northbright/Notes/blob/master/Golang/china/install-go1.6-from-source-on-centos7-in-china.md

https://golang.org/doc/install/source

Go高版本的编译过程需要Go1.4的二进制来实现引导(bootstrap)，简单来说就是：

Go需要Go自身来编译

解决方法：

1.获取Go源码

2.首先编译Go1.4(Go 1.4是C编写的Go工具链的最后一个分发版－官方文档写的。所以它的编译不需要Go编译器，用gcc和glibc-devel。)

3.编译好的Go1.4二进制，来编译Go高版本

步骤：

1.之前已经安装过老版本Go，清除相关环境变量：$GOPATH，$GOROOT

2.安装git // 一般都有

3.安装gcc和glibc-devel // 一般都有

4.下载go源码

cd ~

git clone git@github.com:golang/go.git

cd go

git checkout -b 1.4.3 go1.4.3 // -b 1.4.3 不用也可以，它会创建一个新分支

cd src

./all.bash // 也可简单运行 './make.bash'

编译好的go 1.4.3 版本，默认存储在 ~/go // 在我们执行完 './make.bash' 也有安装目录提示

5.复制 ~/go 到 $GOROOT\_BOOTSTRAP 指定的目录(高版本的Go的构建脚本，该变量值默认是：~/go1.4)

cp -rf ~/go ~/go1.4

6.构建好 go 1.4低版本，我们现在可以开始安装高版本，它借助1.4.3版本的go

cd ~/go

git clean -dfx // 应该是恢复到最初开始，删除掉刚才构建生成的改变

git checkout -b 1.8 go1.8 // 我当前1.8是go的稳定版

cd src

./all.bash

7.高版本安装成功，将Go相关目录，添加到环境变量中

vim ~/.bashrc // 我在mac上用的 zsh，vim ~/.zshrc

export PATH=$PATH:{$HOME}/go/bin

export GOPATH={$HOME}/go-projects

source ~/.bashrc // source ~/.zshrc

简介：

大多数用户，不需要从源码安装go，直接下载二进制包安装即可，非常简单。

官方有两个Go编译器工具链。本文档重点介绍 gc Go编译器及其工具。gccgo编译器是一个更传统的的编译器，使用GCC作为后端。

Go编译器支持8个指令集。

安装二进制Go编译器：

Go工具链是用GO写的。要构建它，需要安装一个Go编译器。初始构件工具的脚本，会在 $GOROOT\_BOOTSTRAP 环境变量中，查找现有的Go工具链。如果未设置，$GOROOT\_BOOTSTRAP 的默认值是 '$HOME/go1.4'。

引导工具链有很多设置选项。获取一个Go工具链后，将 $GOROOT\_BOOTSTRAP 设置为我们获取到的Go工具链的解压目录。例如：

最终 $GOROOT\_BOOTSTRAP/bin/go 应该是引导工具链的go二进制命令。

我们可以在官方的下载页面，下载一个二进制版本作为Go引导工具链，或者使用其他Go分发版。

要从源代码构件Go引导工具链，请使用 git branch release-branch.go1.4或go1.4-bootstrap-20161024.tar.gz，其中包含Go 1.4源代码以及一直以来的修补程序，以保证工具可以运行在较新的操作系统上(Go 1.4是C编写的Go工具链的最后一个分发版)。解压Go 1.4源码包，"cd src/" 子目录，执行 './make.bash'(windows下执行 './make.bat')

从源代码交叉编译Go的引导启动链

1.这对于Go 1.4不定位(target-目标、定位)的系统是很有必要的(例如：linux/ppc64le)

2.用于在一台系统上编译在不同系统的go引导启动链

首先，定义环境变量，然后运行 bootstrap.bash

例如，运行如下命令：

$ GOOS=linux GOARCH=ppc64 ./bootstrap.bash

bootstrap.bash将依据 'GOOS' 和 'GOARCH'，交叉编译Go的工具链，生成到 '../../go-${GOOS}-${GOARCH}-bootstrap'，然后我们可以将适用于不同系统的 Go工具链，复制给指定类型的系统，作为 $GOROOT\_BOOTSTRAP 的路径，来引导本地构建

安装git

一般系统都有，直接yum安装等

安装一个C编译器(可选)

要使用cgo来安装Go，允许Go程序导入C库，必须首先安装C编译器(如：gcc或clang)。使用系统上标准的任何安装方法来执行次操作。

不使用cgo来安装Go，在运行 all.bash 或 make.bash 之前，设置环境变量 CGO\_ENABLED=0

获取go仓库：

git clone git@github.com:golang/go.git

cd go

git checkout go1.8

安装Go：

cd src

./all.bash(windows下使用 ./all.bat)

安装成功，将会输出：

ALL TESTS PASSED

---

Installed Go for linux/amd64 in /home/you/go.

Installed commands in /home/you/go/bin.

\*\*\* You need to add /home/you/go/bin to your $PATH. \*\*\*

最后几行的详细信息反映了安装过程中使用的操作系统、体系结构和根目录。

控制构建方式的更多信息，可参阅下方的环境变量说明。all.bash(all.bat)运行Go相关的重要测试，这可能比简单的建构Go需要花费更多时间。如果不想运行测试套件，使用 make.bash(make.bat) 来替代

测试安装是否成功：

通过构建一个简单的go程序，来检查Go是否正确安装

$ vim hello.go

package main

import "fmt"

func main() {

fmt.Printf("hello, world\n");

}

运行 hello.go

$ go run hello.go

hello world

安装额外工具：

安装所有额外工具：

$ go get golang.org/x/tools/cmd/...

安装指定的一个额外工具：

$ go get golang.org/x/tools/cmd/godoc

注意：

1.go 命令，将会安装 'godoc' 工具到 '$GOROOT/bin'(或者 $GOBIN)

2.cover 和 vet 工具，将被安装到 '$GOROOT/pkg/tool/$GOOS\_$GOARCH'。可以使用 'go tool cover' 和 'go tool vet' 来运行 pkg/tool/ 下安装的命令

环境变量：

可以通过自定义环境变量来改变Go编译环境。构建时，我们可以不设置环境变量，但有时候，可能希望设置其中的一些，来覆盖默认值。

$GOROOT

Go根目录，通常是 "$HOME/go1.x"。它的值，在编译时被设置，默认时 'all.bash' 的上级目录。不需要设置该环境变量，除非有多个go环境(例如：在我们克隆的代码库中，切换不同的版本)

$GOROOT\_FINAL

当$GOROOT未显示设置时，该变量值由已安装的二进制包和脚本来假定。默认同$GOROOT值相同。如果在其他地方构建Go目录，在构建后，将go目录移动到指定目录，将 $GOROOT\_FINAL 设置为最终的目录

$GOOS 和 $GOARCH

操作系统和编译架构的名称。默认同 $GOHOSTOS 和 $GOHOSTARCH 一致。

$GOOS 可以是：

darwin - Mac OS X 10.8+ 以及 iOS

dragonfly

freebsd

linux

netbsd

openbsd

plan9

solaris

windows

$GOARCH 可以是：

amd64 (64-bit x86, the most mature port)

386 (32-bit x86)

arm (32-bit ARM)

arm64 (64-bit ARM)

ppc64le (PowerPC 64-bit, little-endian)

ppc64 (PowerPC 64-bit, big-endian)

mips64le (MIPS 64-bit, little-endian)

and mips64 (MIPS 64-bit, big-endian)

mipsle (MIPS 32-bit, little-endian)

mips (MIPS 32-bit, big-endian)

2者的组合，看文档：

$GOHOSTOS 和 $GOHOSTARCH

主机操作系统的名称和编译架构。默认是：本地系统的操作系统和架构。可选的组合，在上面已经列出(看文档)。指定的值，不许同本地系统兼容。例如：不应该在 'x86' 系统上，设置 $GOHOSTARCH=arm(2个变量的组合必须是上面支持的之一)

$GOBIN

Go二进制软件的安装目录。默认是 $GOROOT/bin。安装完成后，将此目录添加到$PATH环境变量，我们可以直接使用Go相关命令。如果设置了 $GOBIN，go 命令，将会在该目录下，安装所有的命令。(例如：我们通过go安装的额外工具，就会安装到这里)

$GO386

仅限于386，如果建立在386或amd64，则默认为自动检测，否则为387。

该变量来控制gc生成的代码，是使用 387浮点单元(设置为387)，还是SSE2指令(设置为sse2)，来进行浮点计算

GO386 = 387：使用x87进行浮点运算; 应支持所有x86芯片（Pentium MMX或更高版本）。

GO386 = sse2：使用SSE2进行浮点运算; 具有比387更好的性能，但仅适用于Pentium 4 / Opteron / Athlon 64或更高版本。

$GOARM

仅限于arm，如果建立在目标处理器上，默认是自动检测，否则为6

这将设置运行时应该定位的ARM浮点协处理器架构版本。 如果要在目标系统上进行编译，其值将自动检测

GOARM = 5：使用软件浮点数; 当CPU没有VFP协处理器时

GOARM = 6：仅使用VFPv1; 默认如果交叉编译; 通常是ARM11或更好的内核（也支持VFPv2或更好的内核）

GOARM = 7：使用VFPv3; 通常是Cortex-A核心

如果有疑问，请保留此变量未设置，并在首次运行Go可执行文件时根据需要进行调整。 Go社区wiki上的GoARM页面包含有关Go的ARM支持的更多详细信息。

/\*

非常关键，我们可以随意配置$GOARCH和$GOOS，来编译各平台上的go二进制

\*/

注意：

$GOARCH和$GOOS标识目标环境，而非我们机器的运行环境。正因为如此，我们经常使用 "交叉编译"(cross-compiling)。我们可以配置这2个参数，从而编译生成不同平台下的Go二进制，然后复制给其他平台。

如果覆盖默认配置，将这些环境变量设置到shell配置($HOME/.bashrc, $HOME/.profile等)，配置项如下设置：

export GOROOT=$HOME/go1.X

export GOARCH=amd64

export GOOS=linux

再次声明，在构建、安装和开发Go语言包时，并不一定需要设置这些环境变量

————————————————

版权声明：本文为CSDN博主「beyond\_\_devil」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/beyond\_\_devil/article/details/68064202