Introduction

If you want to run your Go programs on a platform that is not supported by the standard Go compiler *gc* you can build a version of the GCC compiler that targets your desired target platform, since GCC supports many more platforms. This is possible because there is a Go frontend to GCC named gccgo.

Details

Definitions

- **Build** This is the computer where you are building the cross-compiler.
- **Host** The computer that will run the cross-compiler once it's built. This is usually the same as Build.
- **Target** This is the destination system, where you want the cross-compiled program to run.

More definitions and complex cross situations can be found at the Wikipedia article.

Go tools and gccgo

You will later need to source code to the Go tool, so you might as well uninstall the version you have installed from your package manager to avoid confusion (fully optional tho). Also build & install gccgo targeting host(!, that's right, you need a gcco compiling for not only target but one for host).

- Installing Go from source
- Official Go documentation

Build the cross-compiler

Build

First you have to build your cross-compiling version of GCC. This is complex process as it requires several stages with bootstrapping since there are mutual dependency relations between GCC and libc implementations. A very fine tutorial on how to build a GCC cross-toolchain with eglic (works with glibc too) was written by Jim Blandy and posted at eglibc's mailinglist patches Cross-building instructions. In the final stage where the full GCC is built, simply configure script with *--enable-languages=c,c++,go* (see official Go documentation).

You can use the ewxb_gcc_cross-compiler_builder script as a starting point. Don't expect that script to work out of the box, but rather as a hint to which steps you're likely to take when building your x-toolchain.

If you're lucky enough and want a version of GCC that is not bleeding edge (which you might want to have the latest Go features) you can use a cross-compiler builder to ease the configuration e.g. crosstool-NG that lets you configure GCC with a simple TUI menu.

Newer versions of crosstool-NG can build the go language by enabling `CT_EXPERIMENTAL` and `CT_CC_SUPPORT_GOLANG`. This will automatically add `go` to `--enable-languages`.

Symlink

You should now have a bin directory with files with names like "`<target>`-gcc", "`<target>`-gnu-gccgo" etc. Because the go build tool does not allow you to specify file-names of the compilers to use (only statically supports the strings 'gc' and 'gccgo') it will look in your $PATH envvar for the first file named 'gccgo' and 'gcc'. You will therefore have to add this directory as an overlay by setting it to be the first one in $PATH when you want to use the

cross-compiler and not your normal gcc binary on your system. Since the go tool looks for a binary named *gccgo* you'll have to make some symlinks for the tools you want it to find.

```
    $ cd path/to/cross-comp-gcc/bin
    $ ln -s <arch>-<os>-gnu-gcc gcc
    $ ln -s <arch>-<os>-gnu-gccgo gccgo
    $ ln -s <arch>-<os>-gnu-ar ar
    $ export PATH="path/to/crosscomp-gcc/bin:$PATH" # Do this whenever you want to use
the cross-compilers targeting your target instead of the system default targeting
host.
```

etc.

With $TARGET set to the target architecture and $PREFIX set to the destination path of the built files, you can make the symlinks with:

```
#!/usr/bin/env bash

cd $PREFIX/bin
for file in $(find . -type f); do
    tool_name=$(echo $file | sed -e "s/${TARGET}-\(.*\)$/\1/")
    ln -sf "$file" "$tool_name"
done
```

Test

When the cross-compiler is build you should test that it works, both for a simple C program and a simple Go program.

```
    $ gccgo -Wall -o helloworld helloworld.go
    $ file helloworld # verify that the architecture of the binary is the desired
target and not a binary that can run on your host machine.
    $ <upload-command-to-target> helloworld
    $ <ssh/telnet etc. to target and test run>
```

Gotchas

If you haven't compiled a shared object of the go library, *libgo*, for your target you might want to compile your Go programs statically, just like *gc* does, to include all what is needed to run your program. Do this by adding the *-static* switch to gccgo. If you're unsure how your produced ELF file is linked, inspect it with *readelf -d* `<elf>` or *objdump -T* `<elf>`.

Build a cross-gccgo aware version of the Go tool

Assumptions

- Assuming that you've followed the instructions on Installing Go from source you should have a checked out version of the go source at *$GOROOT*.
- Envvar *$GOROOT* is set.
- The envvars *$GOARCH* and *$GOOS* represent the **target** architecture and operating system. Figure these out and set them to these values when you want to cross-compile.

Build

You can specify the compiler to use when building with the go tool with *go build -compiler gccgo* `<go-package,files>` . This is however not enough. If you have set *$GOARCH* and *$GOOS* to something that is not supported by *gc* but with *gccgo* you have to build a special version of the go tool that understands these extra architectures. If you compile another version of the go tool with the go tool but specify to use gccgo targeting your host, the resulting go tool will be able to compile programs with all the architectures supported by gccgo.

```
$ cd ~/tmp
$ hg clone ~/src/go # [0]
$ cd go/src/cmd/go
$ go build -o xgo -compiler gccgo . # [1] [2] [3]
$ cp xgo ~/bin/ # Or any directory that is in your $PATH
```

- `[0 ]` Clone Go source to directory outside $GOROOT. Needed since the cmd/go we're about to compile is handles specially.
- `[1 ]` Here the *gccgo* that targets **host** must be used, and not the cross-compiling gccgo. Make sure to have you *$PATH* set correctly for this.
- `[2 ]` Will produce binary named xgo to distinguish from the existing go-binary. Name it to something suitable, e.g. mgo if your target is MIPS.
- `[3 ]` The compilation step (go build) can fail with some compilation errors since gccgo is lagging behind the go project (at least at go mercurial tip 7f2863716967). Fix those errors by commenting out or something more clever and re-compile.

Cross-compile Go programs.

With your *$PATH* set to find _xgo_and the cross-compiling version of_gccgo and *$GOARCH* set properly you can now cross-compile using the go tool (named xgo).

```
$ export PATH="path/to/xgo/:path/to/crosscomp-gccgo/:$PATH"
$ export GOARCH="<you-target's-arch>"
$ export GOOS="<you-target's-OS>"
$ xgo build -compiler gccgo <go-package/files>
```

TODO

Go-programs importing "C" does not seems to work using xgo. Currently blocked by Go issue#7398

This Wiki page was inspired by the lessons learned from "[golang-nuts] Simplification of MIPS cross-compilation?" @ https://groups.google.com/forum/#!topic/golang-nuts/PgyS2yoO2jM