On Configuring Development Tools

K. Richard Pixley, rich@cygnus.com
Cygnus Support

1 Some Basic Terms

There are a lot of terms that are frequently used when discussing development tools. Most of the common terms have been used for many different concepts such that their meanings have become ambiguous to the point of being confusing. Typically, we only guess at their meanings from context and we frequently guess wrong.

This document uses very few terms by comparison. The intent is to make the concepts as clear as possible in order to convey the usage and intent of these tools.

Programs run on *machines*. Programs are very nearly always written in *source*. Programs are *built* from source. *Compilation* is a process that is frequently, but not always, used when building programs.

1.1 Host Environments

In this document, the word *host* refers to the environment in which the source in question will be compiled. *host* and *host name* have nothing to do with the proper name of your host, like *ucbvax*, *prep.ai.mit.edu* or *att.com*. Instead they refer to things like *sun4* and *dec3100*.

Forget for a moment that this particular directory of source is the source for a development environment. Instead, pretend that it is the source for a simpler, more mundane, application, say, a desk calculator.

Source that can be compiled in more than one environment, generally needs to be set up for each environment explicitly. Here we refer to that process as configuration. That is, we configure the source for a host.

For example, if we wanted to configure our mythical desk calculator to compile on a SparcStation, we might configure for host sun4. With our configuration system:

```
cd desk-calculator ; ./configure sun4
```

does the trick. `configure` is a shell script that sets up Makefiles, subdirectories, and symbolic links appropriate for compiling the source on a sun4.

The *host* environment does not necessarily refer to the machine on which the tools are built. It is possible to provide a sun3 development environment on a sun4. If we wanted to use a cross compiler on the sun4 to build a program intended to be run on a sun3, we would configure the source for sun3.

```
cd desk-calculator ; ./configure sun3
```

The fact that we are actually building the program on a sun4 makes no difference if the sun3 cross compiler presents an environment that looks like a sun3 from the point of view of the desk calculator source code. Specifically, the environment is a sun3 environment if the header files, predefined symbols, and libraries appear as they do on a sun3.

Nor does the host environment refer to the the machine on which the program to be built will run. It is possible to provide a sun3 emulation environment on a sun4 such that programs built in a sun3 development environment actually run on the sun4. This technique is often used within individual programs to remedy deficiencies in the host operating system. For example, some operating systems do not provide the `bcopy` function and so it is emulated using the `memcpy` funtion.

Host environment simply refers to the environment in which the program will be built from the source.

1.2 Configuration Time Options

Many programs have compile time options. That is, features of the program that are either compiled into the program or not based on a choice made by the person who builds the program. We refer to these as *configuration options*. For example, our desk calculator might be capable of being compiled into a program that either uses infix notation or postfix as a configuration option. For a sun3, to choose infix you might use:

```
./configure sun3 --enable-notation=infix
```

while for a sun4 with postfix you might use:

```
./configure sun4 --enable-notation=postfix
```

If we wanted to build both at the same time, the intermediate pieces used in the build process must be kept separate.

```
mkdir ../objdir.sun4
(cd ../objdir.sun4 ; ../configure sun4 --enable-notation=postfix --srcdir=../src)
mkdir ../objdir.sun3
(cd ../objdir.sun3 ; ../configure sun3 --enable-notation=infix --srcdir=../src)
```

will create subdirectories for the intermediate pieces of the sun4 and sun3 configurations. This is necessary as previous systems were only capable of one configuration at a time. Otherwise, a second configuration would write over the first. We've chosen to retain this behaviour so the obj directories and the `--srcdir` configuration option are necessary to get the new behaviour. The order of the arguments doesn't matter. There should be exactly one argument without a leading '`-`' and that argument will be assumed to be the host name.

From here on the examples will assume that you want to build the tools *in place* and won't show the `--srcdir` option, but remember that it is available.

In order to actually install the program, the configuration system needs to know where you would like the program installed. The default location is '`/usr/local`'. We refer to this location as `$(prefix)`. All user visible programs will be installed in '`$(prefix)/bin`'. All other programs and files will be installed in a subdirectory of '`$(prefix)/lib`'.

You can only change `$(prefix)` as a configuration time option.

```
./configure sun4 --enable-notation=postfix --prefix=/local
```

Will configure the source such that:

```
make install
```

will put its programs in '`/local/bin`' and '`/local/lib/gcc`'. If you change `$(prefix)` after building the source, you will need to:

```
make clean
```

before the change will be propogated properly. This is because some tools need to know the locations of other tools.

With these concepts in mind, we can drop the desk calculator example and move on to the application that resides in these directories, namely, the source to a development environment.

2 Specifics

The GNU Development Tools can be built on a wide variety of hosts. So, of course, they must be configured. Like the last example,

```
./configure sun4 --prefix=/local
./configure sun3 --prefix=/local
```

will configure the source to be built in subdirectories, in order to keep the intermediate pieces separate, and to be installed in '/local'.

When built with suitable development environments, these will be native tools. We'll explain the term *native* later.

3 Building Development Environments

The GNU development tools can not only be built in a number of host development environments, they can also be configured to create a number of different development environments on each of those hosts. We refer to a specific development environment created as a *target*. That is, the word *target* refers to the development environment produced by compiling this source and installing the resulting programs.

For the GNU development tools, the default target is the same as the host. That is, the development environment produced is intended to be compatible with the environment used to build the tools.

In the example above, we created two configurations, one for sun4 and one for sun3. The first configuration is expecting to be built in a sun4 development environment, to create a sun4 development environment. It doesn't necessarily need to be built on a sun4 if a sun4 development environment is available elsewhere. Likewise, if the available sun4 development environment produces executables intended for something other than sun4, then the development environment built from this sun4 configuration will run on something other than a sun4. From the point of view of the configuration system and the GNU development tools source, this doesn't matter. What matters is that they will be built in a sun4 environment.

Similarly, the second configuration given above is expecting to be built in a sun3 development environment, to create a sun3 development environment.

The development environment produced is a configuration time option, just like `$(prefix)`.

```
./configure sun4 --prefix=/local --target=sun3
./configure sun3 --prefix=/local --target=sun4
```

In this example, like before, we create two configurations. The first is intended to be built in a sun4 environment, in subdirectories, to be installed in '`/local`'. The second is intended to be built in a sun3 environment, in subdirectories, to be installed in '`/local`'.

Unlike the previous example, the first configuration will produce a sun3 development environment, perhaps even suitable for building the second configuration. Likewise, the second configuration will produce a sun4 development environment, perhaps even suitable for building the first configuration.

The development environment used to build these configurations will determine the machines on which the resulting development environments can be used.

4 A Walk Through

4.1 Native Development Environments

Let us assume for a moment that you have a sun4 and that with your sun4 you received a development environment. This development environment is intended to be run on your sun4 to build programs that can be run on your sun4. You could, for instance, run this development environment on your sun4 to build our example desk calculator program. You could then run the desk calculator program on your sun4.

The resulting desk calculator program is referred to as a *native* program. The development environment itself is composed of native programs that, when run, build other native programs. Any other program is referred to as *foreign*. Programs intended for other machines are foreign programs.

This type of development environment, which is by far the most common, is refered to as *native*. That is, a native development environment runs on some machine to build programs for that same machine. The process of using a native development environment to build native programs is called a *native* build.

```
./configure sun4
```

will configure this source such that when built in a sun4 development environment, with a development environment that builds programs intended to be run on sun4 machines, the programs built will be native programs and the resulting development environment will be a native development environment.

The development system that came with your sun4 is one such environment. Using it to build the GNU Development Tools is a very common activity and the resulting development environment is quite popular.

```
make all
```

will build the tools as configured and will assume that you want to use the native development environment that came with your machine.

Using a development environment to build a development environment is called *bootstrapping*. The release of the GNU Development Tools is capable of bootstrapping itself. This is a very powerful feature that we'll return to later. For now, let's pretend that you used the native development environment that came with your sun4 to bootstrap the release and let's call the new development environment *stage1*.

Why bother? Well, most people find that the GNU development environment builds programs that run faster and take up less space than the native development environments that came with their machines. Some people didn't get development environments with their machines and some people just like using the GNU tools better than using other tools.

While you're at it, if the GNU tools produce better programs, maybe you should use them to build the GNU tools. So let's pretend that you do. Let's call the new development environment *stage2*.

So far you've built a development environment, stage1, and you've used stage1 to build a new, faster and smaller development environment, stage2, but you haven't run any of the programs that the GNU tools have built. You really don't yet know if these tools work. Do you have any programs built with the GNU tools? Yes, you do. stage2. What does that program do? It builds programs. Ok, do you have any source handy to build into a program? Yes, you do. The GNU tools themselves. In fact, if you use stage2 to build the GNU tools again the resulting programs should be identical to stage2. Let's pretend that you do and call the new development environment *stage3*.

You've just completed what's called a *three stage boot*. You now have a small, fast, somewhat tested, development environment.

```
make bootstrap
```

will do a three stage boot across all tools and will compare stage2 to stage3 and complain if they are not identical.

Once built,

```
make install
```

will install the development environment in the default location, or in `$(prefix)` if you specified an alternate when you configured.

Any development environment that is not a native development environment is refered to as a *cross* development environment. There are many different types of cross development environments but most fall into one of three basic categories.

4.2 Emulation Environments

The first category of cross development environment is called *emulation*. There are two primary types of emulation, but both types result in programs that run on the native host.

The first type is *software emulation*. This form of cross development environment involves a native program that when run on the native host, is capable of interpreting, and in most aspects running, a program intended for some other machine. This technique is typically used when the other machine is either too expensive, too slow, too fast, or not available, perhaps because it hasn't yet been built. The native, interpreting program is called a *software emulator*.

The GNU Development Tools do not currently include any software emulators. Some do exist and the GNU Development Tools can be configured to create simple cross development environments for with these emulators. More on this later.

The second type of emulation is when source intended for some other development environment is built into a program intended for the native host. The concepts of operating system universes and hosted operating systems are two such development environments.

4.3 Simple Cross Environments

```
./configure sun4 --target=a29k
```

will configure the tools such that when compiled in a sun4 development environment the resulting development environment can be used to create programs intended for an a29k. Again, this does not necessarily mean that the new development environment can be run on a sun4. That would depend on the development environment used to build these tools.

Earlier you saw how to configure the tools to build a native development environment, that is, a development environment that runs on your sun4 and builds programs for your sun4. Let's pretend that you use stage3 to build this simple cross configuration and let's call the new development environment gcc-a29k. Remember that this is a native build. Gcc-a29k is a collection of native programs intended to run on your sun4. That's what stage3 builds, programs for your sun4. Gcc-a29k represents an a29k development environment that builds programs intended to run on an a29k. But, remember, gcc-a29k runs on your sun4. Programs built with gcc-a29k will run on your sun4 only with the help of an appropriate software emulator.

Building gcc-a29k is also a bootstrap but of a slightly different sort. We call gcc-a29k a *simple cross* environment and using gcc-a29k to build a program intended for a29k is called *crossing to* a29k. Simple cross environments are the second category of cross development environments.

4.4 Crossing Into Targets

```
./configure a29k --target=a29k
```

will configure the tools such that when compiled in an a29k development environment, the resulting development environment can be used to create programs intended for an a29k. Again, this does not necessarily mean that the new development environment can be run on an a29k. That would depend on the development environment used to build these tools.

If you've been following along this walk through, then you've already built an a29k environment, namely gcc-a29k. Let's pretend you use gcc-a29k to build the current configuration.

Gcc-a29k builds programs intended for the a29k so the new development environment will be intended for use on an a29k. That is, this new gcc consists of programs that are foreign to your sun4. They cannot be run on your sun4.

The process of building this configuration is a another bootstrap. This bootstrap is also a cross to a29k. Because this type of build is both a bootstrap and a cross to a29k, it is sometimes referred to as a *cross into* a29k. This new development environment isn't really a cross development environment at all. It is intended to run on an a29k to produce programs for an a29k. You'll remember that this makes it, by definition, an a29k native compiler. *Crossing into* has been introduced here not because it is a type of cross development environment, but because it is frequently mistaken as one. The process is a *cross* but the resulting development environment is a native development environment.

You could not have built this configuration with stage3, because stage3 doesn't provide an a29k environment. Instead it provides a sun4 environment.

If you happen to have an a29k lying around, you could now use this fresh development environment on the a29k to three-stage these tools all over again. This process would look just like it did when we built the native sun4 development environment because we would be building another native development environment, this one on a29k.

4.5 Canadian Cross

So far you've seen that our development environment source must be configured for a specific host and for a specific target. You've also seen that the resulting development environment depends on the development environment used in the build process.

When all four match identically, that is, the configured host, the configured target, the environment presented by the development environment used in the build, and the machine on which the resulting development environment is intended to run, then the new development environment will be a native development environment.

When all four match except the configured host, then we can assume that the development environment used in the build is some form of library emulation.

When all four match except for the configured target, then the resulting development environment will be a simple cross development environment.

When all four match except for the host on which the development environment used in the build runs, the build process is a *cross into* and the resulting development environment will be native to some other machine.

Most of the other permutations do exist in some form, but only one more is interesting to the current discussion.

```
./configure a29k --target=sun3
```

will configure the tools such that when compiled in an a29k development environment, the resulting development environment can be used to create programs intended for a sun3. Again, this does not necessarily mean that the new development environment can be run on an a29k. That would depend on the development environment used to build these tools.

If you are still following along, then you have two a29k development environments, the native development environment that runs on a29k, and the simple cross that runs on your sun4. If you use the a29k native development environment on the a29k, you will be doing the same thing we did a while back, namely building a simple cross from a29k to sun3. Let's pretend that instead, you use gcc-a29k, the simple cross development environment that runs on sun4 but produces programs for a29k.

The resulting development environment will run on a29k because that's what gcc-a29k builds, a29k programs. This development environment will produce programs for a sun3 because that is how it was configured. This means that the resulting development environment is a simple cross.

There really isn't a common name for this process because very few development environments are capable of being configured this extensively. For the sake of discussion, let's call this process a *Canadian cross*. It's a three party cross, Canada has a three party system, hence Canadian Cross.

5 Final Notes

By *configures*, I mean that links, Makefile, .gdbinit, and config.status are built. Configuration is always done from the source directory.

`./configure` *name*
: configures this directory, perhaps recursively, for a single host+target pair where the host and target are both *name*. If a previous configuration existed, it will be overwritten.

`./configure` *hostname* `--target=`*targetname*
: configures this directory, perhaps recursively, for a single host+target pair where the host is *hostname* and target is *targetname*. If a previous configuration existed, it will be overwritten.

5.1 Hacking Configurations

The configure scripts essentially do three things, create subdirectories if appropriate, build a 'Makefile', and create links to files, all based on and tailored to, a specific host+target pair. The scripts also create a '.gdbinit' if appropriate but this is not tailored.

The Makefile is created by prepending some variable definitions to a Makefile template called 'Makefile.in' and then inserting host and target specific Makefile fragments. The variables are set based on the chosen host+target pair and build style, that is, if you use `--srcdir` or not. The host and target specific Makefile may or may not exist.

- Makefiles can be edited directly, but those changes will eventually be lost. Changes intended to be permanent for a specific host should be made to the host specific Makefile fragment. This should be in './config/mh-*host*' if it exists. Changes intended to be permanent for a specific target should be made to the target specific Makefile fragment. This should be in './config/mt-*target*' if it exists. Changes intended to be permanent for the directory should be made in 'Makefile.in'. To propogate changes to any of these, either use `make Makefile` or `./config.status` or re-configure.

Appendix A Index

B

C

E

F

H

M

N

P

S

T

Table of Contents