
Cross-compilation

Cross Compilation: GNU/Linux Conventions

A cross-compiler is a compiler that creates executable code for another processor

Building by yourself cross-code requires three parameters

- ♦ **--build** (where is the code being built -- usually autodetected)
- ♦ **--host** (where will the code be hosted)
- ♦ **--target** (where will the code run)

Cross-compilation tools usually have a prefix in the filename

- ♦ **arm-linux-gcc arm-buildroot-linux-gnueabi-gcc**
- ♦ **m68k-linux-gcc lm32-elf-gcc**

Building the kernel:

```
make CROSS_COMPILE=m68k-linux- ARCH=m68knommu
```

Creating a Cross Compiler

To cross-compile, you need three packages

- **binutils (assembler, linker)**
- **gcc (the compiler proper)**
- **glibc or another libc implementation**
 - ◆ **As a minimal fallback, you can use "newlib"**

The steps to build the compiler are:

- **compiling binutils**
- **compiling the bootstrap gcc**
- **compiling libc**
- **compiling the final gcc (with C++, Java etc)**

Most embedded distributions nowadays build the compiler first

Still, you can build your own, especially for uC targets

Pre-Built and Crosstool-ng

A few developers or companies offer pre-built toolchains

- Unfortunately, the list is very volatile
- Refer to elinux for a "current" list
 - ◆ <https://elinux.org/Toolchains>

Another option is crosstool-ng

- Much more difficult than the original "crosstool"
- Kconfig based
- Reported to be reliable

For bare-metal (or kernel, or bootloader), distributions help

- A suitable arm-none-eabi-gcc is usually packaged
- Check your distribution for details

Please note the naming: "arm-none-eabi"

- The first word is the CPU family
- The second word is the host operating system
- The later (optional) words are variants
- We'll talk about ARM and EABI later on

Programs Included in Binutils

Tool-chain proper commands

- **as**
- **ld**

Maintenance commands

- **strip**
- **objcopy**

Information retrieval commands

- **objdump, nm**
- **size, strings**

Most of the information is abstracted to libbfd

- **The library offers An API to read/write an object file**
- **You can create a multi-platform objcopy**
- **Some distributions offer "binutils-multiarch"**
 - ◆ **It lacks AVR or LM32 support, but most CPU families are there**

Cross Compiling Applications

Simple packages:

```
make CC=my-cross-gcc
```

Autotools packages:

```
CC=my-cross-gcc ./configure --prefix=/usr --host=arm-linux \  
    && make && make install DESTDIR=/target
```

Special case: gdbserver (tested w/ version 9.1):

```
CC=my-cross-gcc /path/to/gdb-9.1/gdb/gdbserver/configure \  
    --host=arm-linux --prefix=/usr \  
    && make && make install DESTDIR=/target
```

Special case: gdb (tested w/version 9.1):

```
/path/to/gdb-9.1/configure --target=arm-linux \  
    --prefix=/usr/local/cross-tools \  
    && make && make install
```

Cross Compiling Bare-Metal systems

Most bare-metal systems follow the Linux tradition:

- The "CROSS_COMPILE" variable sets the prefix
- The "ARCH" variable sets the top-level subtree

Also, most bare-metal systems are based on Kconfig

- You can "make defconfig" (depends on \$ARCH)
- Mode defaults live in the configs/ subdir
- You can interactively change the configuration
 - ♦ "make config", "make oldconfig", "make menuconfig", ...
- The current configuration lives in .config (the output of configuring)
 - ♦ This is an input file for make
 - ♦ A corresponding header is generated for C language

One problem with Kconfig is reproducing a build

- If it can fit, it makes sense to save .config in the binary
- This in addition to the build commit, a mandatory item

Please consider setting up reproducible builds for your own OS