How to Build an iOS Toolchain for Linux (Debian 7)

By Roger Keays, 4 April 2015

After jailbreaking an iPhone I discovered it was a device that had more behind it than just a dumbed down user interface. I had unlocked a unix operating system with a complete package manager (cydia/apt) and development tools to boot. This was pretty cool, I could even compile programs for iPhone on an iPhone.

Cool, yes. But perhaps not very practical for developing complex applications. For that you need a mac right?

Well... no. You don't.

Look at the tools under the hood and you'll see it is a mostly open source stack powered by clang, llvm and a custom linker for the darwin/mach kernel. This article will show you how to build an iOS toolchain for Linux Debian 7 (Wheezy). It is basically a fork of the instructions on the iOS Clang Toolchain installation page but has been updated for Debian 7 and includes a few little patches and tweaks here and there.

Step 1. Install Clang

Clang and llvm compile C, C++ and Objective-C to machine code. They support ARM targets out of the box so we use the upstream versions. We're using the 3.4 version of llvm because some issues have been reported with the 3.5 version.

- remove any old llvm versions
  ```
  # apt-get remove llvm-* libllvm-*
  ```
- add the llvm-3.4 repos to apt
  ```
  # echo 'deb http://llvm.org/apt/wheezy/ llvm-toolchain-wheezy-3.4-binaries main' >> /etc/apt/sources.list
  # wget -O - http://llvm.org/apt/llvm-snapshot.gpg.key | sudo apt-key add -
  # apt-get update
  ```

- install clang and dependencies
  ```
  # apt-get install clang-3.4 libclang-3.4-dev llvm-3 4 libtool bison
  flex automake
  ```
- add llvm-3.4 libraries to the linker library path
  ```
  # echo '/usr/lib/llvm-3.4/lib' > /etc/ld.so.conf.d/libllvm-3.4.conf
  # ldconfig
  ```
- create symlinks for llvm headers and config command
  ```
  # ln -s /usr/lib/llvm-3.4/bin/llvm-config /usr/bin/llvm-config
  # ln -s /usr/include/llvm-3.4/llvm /usr/include/llvm
  # ln -s /usr/include/llvm-c-3.4/llvm-c /usr/include/llvm-c
  ```

Step 2. Build the linker (cctools)

The GNU linker cannot create binaries for iOS, so we have to build the Apple linker which is made easier by the fork maintained in the toolchain project.

- checkout the cctools project
  ```
  # svn checkout http://ios-toolchain-based-on-clang-for-linux.googlecode.
  com/svn/trunk/cctools-porting
  ```
- edit and patch cctools-ld64.sh to use wget instead of proz
  ```
  #proz -k=20 --no-curses $LD64_URL
  wget $LD64_URL
  ```
- checkout and patch the linker source
  ```
  # ./cctools-ld64.sh
  ```
- build the compiler tools
  ```
  # cd cctools-855-ld64-236.3
  # ./autogen.sh
  # ./configure --prefix=/usr/local --target=arm-apple-darwin11
  # make
  # make install
  ```
- create a symlink the new linker
  ```
  # mv /usr/bin/ld /usr/bin/ld.old
  # ln -s /usr/local/bin/arm-apple-darwin11-ld /usr/bin/ld
  ```

Note this installs a new linker (ld) in /usr/bin. If you're building C apps for your local system you'll have to restore the old linker by reversing those last two commands. The two linkers are not compatible.

Step 3. Install ios-tools

These tools come from the toolchain project (again) and include utilities to make builds easier such as converting xcode projects to Makefiles and configuring the location of the SDK.

- download iphonesdk-utils

  ```
  # wget https://ios-toolchain-based-on-clang-for-linux.googlecode.com
  /files/iphonesdk-utils-2.0.tar.gz
  # tar -zxf iphonesdk-utils-2.0.tar.gz
  ```

- or checkout from svn

  ```
  # svn co http://ios-toolchain-based-on-clang-for-linux.googlecode.com
  /svn/trunk/iphonesdk-utils
  ```

- patch genLocalization2/getLocalizedStringFromFile.cpp

```
*** getLocalizedStringFromFile.cpp~ 2015-04-02 04:45:39.309837816 +0530
--- getLocalizedStringFromFile.cpp 2015-04-02 04:45:11.525700021 +0530
***************
*** 113,115 ****
 clang::HeaderSearch headerSearch(headerSearchOptions,
- fileManager,
 *pDiagnosticsEngine,
--- 113,115 ----
 clang::HeaderSearch headerSearch(headerSearchOptions,
+ sourceManager,
 *pDiagnosticsEngine,
***************
*** 129,134 ****
 false);
- clang::HeaderSearch headerSearch(fileManager,
 *pDiagnosticsEngine,
 languageOptions,
- pTargetInfo);
 ApplyHeaderSearchOptions(headerSearch, headerSearchOptions, languageOptions, p
--- 129,134 ----
 false);
+ clang::HeaderSearch headerSearch(fileManager);/*,
 *pDiagnosticsEngine,
 languageOptions,
+ pTargetInfo);*/
 ApplyHeaderSearchOptions(headerSearch, headerSearchOptions, languageOptions, p
```

- build iphonesdk-utils

  ```
  # ./autogen.sh
  ```

```
# ./configure --prefix=/usr/local
# make
# make install
```

The patch above is just to get it to compile and may break something in that utility (**ios-genLocalization**). It is necessary because the llvm API has been changing frequently.

Step 4. Install the SDK

You're now ready to build apps for iOS, but chances are you will be wanting to use libraries from the iOS SDK, so you might as well download it now. The SDK ships with xcode and is difficult to extract (thanks Apple!) but there are some pre-extracted versions available online.

- option 1: download and extract from http://iphone.howett.net/sdks/
  ```
  # wget http://iphone.howett.net/sdks/dl/iPhoneOS8.1.sdk.tbz2
  # tar -xjf iPhoneOS8.1.sdk.tbz2
  ```
- option 2: download and extract from http://code.google.com/p/ios-toolchain-based-on-clang-for-linux/downloads/list
  ```
  # apt-get install xz-utils
  # wget https://ios-toolchain-based-on-clang-for-linux.googlecode.com
  /files/iPhoneOS6.0.sdk.tar.xz
  # tar -xJf iPhoneOS6.0.sdk.tar.xz
  ```

Step 5. Hello World!

ios-createProject includes a sample HelloWorld app you can build with your new tools. If all goes well, it'll be running on your iPhone in a couple of minutes :) You will need OpenSSH installed on your iPhone, which can be installed with cydia or apt-get.

- create helloworld project
  ```
  # ios-createProject
  Choose a Template: 0 (Application)
  Select a Project Name: HelloWorld
  ```
- build and deploy
  ```
  # make
  # make install IPHONE_IP=<your iphone ip>
  ```

Have fun, let me know how you go in the comments below.

Troubleshooting

You're a software developer, you're used to this shite right? Your first port of call is probably the GitHub issues for the toolchain project. If you post some patches in the comments below though I'll add them here to help people along a bit.

```
# ios-clang -c -objc-arc -fblocks -g0 -O2 -I"." HelloWorldApplication.m -o Hel
terminate called after throwing an instance of 'std::out_of_range'
 what(): basic_string::substr
```

The above error is caused by ios-clang being unable to find the cctools binaries. Make sure you have `/usr/local/bin` on your PATH and check the binaries have been installed (they all start with arm-apple-darwin11.

```
invalid option for ld: -dynamic
```

This error occurs when you try to link using the GNU linker (or some other linker). Make sure you symlink `/usr/local/bin/arm-apple-darwin11-ld` to `/usr/bin/ld`.

About Roger Keays

Roger Keays is an artist, an engineer, and a student of life. He has no fixed addressand has leftfootprints on 40-something different countries around the world. Roger is addicted to surfing. His other interests are music, psychology, languages, the proper use of semicolons, and finding good food.