NAME

perlandroid - Perl under Android

SYNOPSIS

The first portions of this documents contains instructions to cross-compile Perl for Android 2.0 and later, using the binaries provided by Google. The latter portion describes how to build perl native using one of the toolchains available on the Play Store.

DESCRIPTION

This document describes how to set up your host environment when attempting to build Perl for Android.

Cross-compilation

These instructions assume an Unixish build environment on your host system; they've been tested on Linux and OS X, and may work on Cygwin and MSYS. While Google also provides an NDK for Windows, these steps won't work native there, although it may be possible to cross-compile through different means.

If your host system's architecture is 32 bits, remember to change the `x86_64`'s below to `x86`'s. On a similar vein, the examples below use the 4.8 toolchain; if you want to use something older or newer (for example, the 4.4.3 toolchain included in the 8th revision of the NDK), just change those to the relevant version.

Get the Android Native Development Kit (NDK)

You can download the NDK from <https://developer.android.com/tools/sdk/ndk/index.html>. You'll want the normal, non-legacy version.

Determine the architecture you'll be cross-compiling for

There's three possible options: arm-linux-androideabi for ARM, mipsel-linux-android for MIPS, and simply x86 for x86. As of 2014, most Android devices run on ARM, so that is generally a safe bet.

With those two in hand, you should add

```
$ANDROID_NDK/toolchains/$TARGETARCH-4.8/prebuilt/`uname | tr '[A-Z]' '[a-z]'`-x8
```

to your `PATH`, where `$ANDROID_NDK` is the location where you unpacked the NDK, and `$TARGETARCH` is your target's architecture.

Set up a standalone toolchain

This creates a working sysroot that we can feed to Configure later.

```
$ export ANDROID_TOOLCHAIN=/tmp/my-toolchain-$TARGETARCH
$ export SYSROOT=$ANDROID_TOOLCHAIN/sysroot
$ $ANDROID_NDK/build/tools/make-standalone-toolchain.sh \
--platform=android-9 \
--install-dir=$ANDROID_TOOLCHAIN \
--system=`uname | tr '[A-Z]' '[a-z]'`-x86_64 \
--toolchain=$TARGETARCH-4.8
```

adb or ssh?

adb is the Android Debug Bridge. For our purposes, it's basically a way of establishing an ssh connection to an Android device without having to install anything on the device itself, as long as the device is either on the same local network as the host, or it is connected to the host through USB.

Perl can be cross-compiled using either adb or a normal ssh connection; in general, if you can connect your device to the host using a USB port, or if you don't feel like installing an sshd app on your device, you may want to use adb, although you may be forced to switch to ssh if your device is not rooted and you're unlucky — more on that later. Alternatively, if you're cross-compiling to an emulator, you'll have to use adb.

adb

To use adb, download the Android SDK from <https://developer.android.com/sdk/index.html>. The ''SDK Tools Only'' version should suffice — if you downloaded the ADT Bundle, you can find the sdk under *$ADT_BUNDLE/sdk/*.

Add *$ANDROID_SDK/platform-tools* to your PATH, which should give you access to adb. You'll now have to find your device's name using `adb devices`, and later pass that to Configure through `-Dtargethost=$DEVICE`.

However, before calling Configure, you need to check if using adb is a viable choice in the first place. Because Android doesn't have a */tmp*, nor does it allow executables in the sdcard, we need to find somewhere in the device for Configure to put some files in, as well as for the tests to run in. If your device is rooted, then you're good. Try running these:

```
$ export TARGETDIR=/mnt/asec/perl
$ adb -s $DEVICE shell "echo sh -c '\"mkdir $TARGETDIR\"' | su --"
```

Which will create the directory we need, and you can move on to the next step. */mnt/asec* is mounted as a tmpfs in Android, but it's only accessible to root.

If your device is not rooted, you may still be in luck. Try running this:

```
$ export TARGETDIR=/data/local/tmp/perl
$ adb -s $DEVICE shell "mkdir $TARGETDIR"
```

If the command works, you can move to the next step, but beware: **You'll have to remove the directory from the device once you are done! Unlike */mnt/asec*, */data/local/tmp* may not get automatically garbage collected once you shut off the phone**.

If neither of those work, then you can't use adb to cross-compile to your device. Either try rooting it, or go for the ssh route.

ssh

To use ssh, you'll need to install and run a sshd app and set it up properly. There are several paid and free apps that do this rather easily, so you should be able to spot one on the store. Remember that Perl requires a passwordless connection, so set up a public key.

Note that several apps spew crap to stderr every time you connect, which can throw off Configure. You may need to monkeypatch the part of Configure that creates `run-ssh` to have it discard stderr.

Since you're using ssh, you'll have to pass some extra arguments to Configure:

```
-Dtargetrun=ssh -Dtargethost=$TARGETHOST -Dtargetuser=$TARGETUSER -Dtargetport=$
```

Configure and beyond

With all of the previous done, you're now ready to call Configure.

If using adb, a ''basic'' Configure line will look like this:

```
$ ./Configure -des -Dusedevel -Dusecrosscompile -Dtargetrun=adb \
-Dcc=$TARGETARCH-gcc \
-Dsysroot=$SYSROOT \
-Dtargetdir=$TARGETDIR \
-Dtargethost=$DEVICE
```

If using ssh, it's not too different — we just change targetrun to ssh, and pass in targetuser and targetport. It ends up looking like this:

```
$ ./Configure -des -Dusedevel -Dusecrosscompile -Dtargetrun=ssh \
-Dcc=$TARGETARCH-gcc \
-Dsysroot=$SYSROOT \
-Dtargetdir=$TARGETDIR \
-Dtargethost="$TARGETHOST" \
-Dtargetuser=$TARGETUSER \
-Dtargetport=$TARGETPORT
```

Now you're ready to run `make` and `make test`!

As a final word of warning, if you're using adb, `make test` may appear to hang; this is because it doesn't output anything until it finishes running all tests. You can check its progress by logging into the device,

moving to *$TARGETDIR*, and looking at the file *output.stdout*.

Notes

- If you are targetting x86 Android, you will have to change `$TARGETARCH-gcc` to `i686-linux-android-gcc`.
- On some older low-end devices — think early 2.2 era — some tests, particularly*t/r e/uniprops.t*, may crash the phone, causing it to turn itself off once, and then back on again.

Native Builds

While Google doesn't provide a native toolchain for Android, you can still get one from the Play Store; for example, there's the CCTools app which you can get for free. Keep in mind that you want a full toolchain; some apps tend to default to installing only a barebones version without some important utilities, like ar or nm.

Once you have the toolchain set up properly, the only remaining hurdle is actually locating where in the device it was installed in. For example, CCTools installs its toolchain in */data/data/com.pdaxrom.cctools/root/cctools*. With the path in hand, compiling perl is little more than:

```
export SYSROOT=<location of the native toolchain>
export LD_LIBRARY_PATH="$SYSROOT/lib:`pwd`:`pwd`/lib:`pwd`/lib/auto:$LD_LIBRARY_
sh Configure -des -Dsysroot=$SYSROOT -Alibpth="/system/lib /vendor/lib"
```

AUTHOR

Brian Fraser <fraserbn@gmail.com>