Notes for Android platforms

Requirement details ——————-

Beside basic tools like perl and make you'll need to download the Android NDK. It's available for Linux, macOS and Windows, but only Linux version was actually tested. There is no reason to believe that macOS wouldn't work. And as for Windows, it's unclear which "shell" would be suitable, MSYS2 might have best chances. NDK version should play lesser role, the goal is to support a range of most recent versions.

Configuration ————-

Android is a cross-compiled target and you can't rely on `./Configure` to find out the configuration target for you. You have to name your target explicitly; there are `android-arm`, `android-arm64`, `android-mips`, `android-mip64`, `android-x86` and `android-x86_64` (`*MIPS` targets are no longer supported with NDK R20+).

Do not pass –cross-compile-prefix (as you might be tempted), as it will be "calculated" automatically based on chosen platform. However, you still need to know the prefix to extend your PATH, in order to invoke `$(CROSS_COMPILE)clang` [`*gcc` on NDK 19 and lower] and company. (`./Configure` will fail and give you a hint if you get it wrong.)

Apart from `PATH` adjustment you need to set `ANDROID_NDK_ROOT` environment to point at the `NDK` directory. If you're using a side-by-side NDK the path will look something like `/some/where/android-sdk/ndk/<ver>`, and for a standalone NDK the path will be something like `/some/where/android-ndk-<ver>`. Both variables are significant at both configuration and compilation times. The NDK customarily supports multiple Android API levels, e.g. `android-14`, `android-21`, etc. By default latest API level is chosen. If you need to target an older platform pass the argument `-D__ANDROID_API__=N` to `Configure`, with `N` being the numerical value of the target platform version. For example, to compile for Android 10 arm64 with a side-by-side NDK r20.0.5594570

```
export ANDROID_NDK_ROOT=/home/whoever/Android/android-sdk/ndk/20.0.5594570
PATH=$ANDROID_NDK_ROOT/toolchains/llvm/prebuilt/linux-x86_64/bin:$ANDROID_NDK_ROOT/toolchain
./Configure android-arm64 -D__ANDROID_API__=29
make
```

Older versions of the NDK have GCC under their common prebuilt tools directory, so the bin path will be slightly different. EG: to compile for ICS on ARM with NDK 10d:

```
export ANDROID_NDK_ROOT=/some/where/android-ndk-10d
PATH=$ANDROID_NDK_ROOT/toolchains/arm-linux-androideabi-4.8/prebuilt/linux-x86_64/bin:$PATH
./Configure android-arm -D__ANDROID_API__=14
make
```

Caveat lector! Earlier OpenSSL versions relied on additional `CROSS_SYSROOT` variable set to `$ANDROID_NDK_ROOT/platforms/android-<api>/arch-<arch>` to appoint headers-n-libraries' location. It's still recognized in order to facilitate migration from older projects. However, since API level appears in `CROSS_SYSROOT` value, passing `-D__ANDROID_API__=N` can be in conflict, and mixing the two is therefore not supported. Migration to `CROSS_SYSROOT`-less setup is recommended.

One can engage clang by adjusting PATH to cover same NDK's clang. Just keep in mind that if you miss it, Configure will try to use gcc... Also, PATH would need even further adjustment to cover unprefixed, yet target-specific, ar and ranlib. It's possible that you don't need to bother, if binutils-multiarch is installed on your Linux system.

Another option is to create so called "standalone toolchain" tailored for single specific platform including Android API level, and assign its location to `ANDROID_NDK_ROOT`. In such case you have to pass matching target name to Configure and shouldn't use `-D__ANDROID_API__=N`. `PATH` adjustment becomes simpler, `$ANDROID_NDK_ROOT/bin:$PATH` suffices.

Running tests (on Linux) ——————

This is not actually supported. Notes are meant rather as inspiration.

Even though build output targets alien system, it's possible to execute test suite on Linux system by employing qemu-user. The trick is static linking. Pass -static to Configure, then edit generated Makefile and remove occurrences of -ldl and -pie flags. You would also need to pick API version that comes with usable static libraries, 42/2=21 used to work. Once built, you should be able to

```
env EXE_SHELL=qemu-<arch> make test
```

If you need to pass additional flag to qemu, quotes are your friend, e.g.

```
env EXE_SHELL="qemu-mips64el -cpu MIPS64R6-generic" make test
```