Notes for UNIX-like platforms

For Unix/POSIX runtime systems on Windows, please see the Notes for Windows platforms.

OpenSSL uses the compiler to link programs and shared libraries

OpenSSL's generated Makefile uses the C compiler command line to link programs, shared libraries and dynamically loadable shared objects. Because of this, any linking option that's given to the configuration scripts MUST be in a form that the compiler can accept. This varies between systems, where some have compilers that accept linker flags directly, while others take them in `-Wl,` form. You need to read your compiler documentation to figure out what is acceptable, and `ld(1)` to figure out what linker options are available.

Shared libraries and installation in non-default locations

Every Unix system has its own set of default locations for shared libraries, such as `/lib`, `/usr/lib` or possibly `/usr/local/lib`. If libraries are installed in non-default locations, dynamically linked binaries will not find them and therefore fail to run, unless they get a bit of help from a defined runtime shared library search path.

For OpenSSL's application (the `openssl` command), our configuration scripts do NOT generally set the runtime shared library search path for you. It's therefore advisable to set it explicitly when configuring, unless the libraries are to be installed in directories that you know to be in the default list.

Runtime shared library search paths are specified with different linking options depending on operating system and versions thereof, and are talked about differently in their respective documentation; variations of RPATH are the most usual (note: ELF systems have two such tags, more on that below).

Possible options to set the runtime shared library search path include the following:

```
-Wl,-rpath,/whatever/path   # Linux, *BSD, etc.
-R /whatever/path           # Solaris
-Wl,-R,/whatever/path       # AIX (-bsvr4 is passed internally)
-Wl,+b,/whatever/path       # HP-UX
-rpath /whatever/path       # Tru64, IRIX
```

OpenSSL's configuration scripts recognise all these options and pass them to the Makefile that they build. (In fact, all arguments starting with `-Wl,` are recognised as linker options.) Please note that 'l' in '-Wl' is lowercase L and not 1.

Please do not use verbatim directories in your runtime shared library search path! Some OpenSSL config targets add an extra directory level for multilib installations. To help with that, the produced Makefile includes the variable LIBRPATH, which is a convenience variable to be used with the runtime shared library search path options, as shown in this example:

```
$ ./Configure --prefix=/usr/local/ssl --openssldir=/usr/local/ssl \
    '-Wl,-rpath,$(LIBRPATH)'
```

On modern ELF based systems, there are two runtime search paths tags to consider, `DT_RPATH` and `DT_RUNPATH`. Shared objects are searched for in this order:

1. Using directories specified in DT_RPATH, unless DT_RUNPATH is also set.
2. Using the environment variable LD_LIBRARY_PATH
3. Using directories specified in DT_RUNPATH.
4. Using system shared object caches and default directories.

This means that the values in the environment variable `LD_LIBRARY_PATH` won't matter if the library is found in the paths given by `DT_RPATH` (and `DT_RUNPATH` isn't set).

Exactly which of `DT_RPATH` or `DT_RUNPATH` is set by default appears to depend on the system. For example, according to documentation, `DT_RPATH` appears to be deprecated on Solaris in favor of `DT_RUNPATH`, while on Debian GNU/Linux, either can be set, and `DT_RPATH` is the default at the time of writing.

How to choose which runtime search path tag is to be set depends on your system, please refer to ld(1) for the exact information on your system. As an example, the way to ensure the `DT_RUNPATH` is set on Debian GNU/Linux systems rather than DT_RPATH is to tell the linker to set new dtags, like this:

```
$ ./Configure --prefix=/usr/local/ssl --openssldir=/usr/local/ssl \
    '-Wl,--enable-new-dtags,-rpath,$(LIBRPATH)'
```

It might be worth noting that some/most ELF systems implement support for runtime search path relative to the directory containing current executable, by interpreting `$ORIGIN` along with some other internal variables. Consult your system documentation.

Linking your application

Third-party applications dynamically linked with OpenSSL (or any other) shared library face exactly the same problem with non-default locations. The OpenSSL config options mentioned above might or might not have bearing on linking of the target application. "Might" means that under some circumstances it would be sufficient to link with OpenSSL shared library "naturally", i.e. with `-L/whatever/path -lssl -lcrypto`. But there are also cases when you'd have to explicitly specify runtime search path when linking your application. Consult your system documentation and use above section as inspiration...

Shared OpenSSL builds also install static libraries. Linking with the latter is likely to require special care, because linkers usually look for shared libraries first and tend to remain "blind" to static OpenSSL libraries. Referring to system documentation would suffice, if not for a corner case. On AIX static libraries (in shared build) are named differently, add `_a` suffix to link with them, e.g. `-lcrypto_a`.