Linking to Libraries

coursera.org/learn/linux-tools-for-developers/supplement/2bPzq/linking-to-libraries

Whether a library is static or fixed, executables are linked to the library with:

```
$ gcc -o foo foo.c -L/mypath/lib -lfoolib
```

This will link in **/mypath/lib/libfoolib.so**, if it exists, and **/mypath/lib/libfoolib.a** otherwise.

The name convention is such that **-lwhat** refers to library **libwhat.so(.a)**. If both **libwhat.so** and **libwhat.a** exist, the shared library will be used, unless **-static** is used on the compile line.

Poorly designed projects may have circular library dependencies, and since the loader makes only one pass through the libraries requested, you can have something like:

```
$ gcc .... -lA -lB -lA ..
```

if **libB** refers to a symbol in **libA** which is not otherwise referred to. While there is an option to make the loader make multiple passes (see info **gcc** for details) it is very slow, and a proper, layered, library architecture should avoid this kind of going in circles.

The default library search path will always include **/usr/lib** and **/lib**. To see exactly what is being searched you can do **gcc --print-search-dirs**. User-specified library paths come before the default ones, although there are extended options to **gcc** to reverse this pattern.

Stripping executables

The command:

```
$ strip foobar
```

where **foobar** is an executable program, object file, or library archive, can be used to reduce file size and save disk space. The symbol table is discarded. This step is generally done on production versions.

Do not use **strip** on either the Linux kernel or kernel modules, both of which need the symbol information!

Getting Debug Information

You can use the environment variable **LD_DEBUG** to obtain useful debugging information. For instance, doing:

```
$ LD_DEBUG=help
```

and then typing any command gives:

```
$ ls
Valid options for the LD_DEBUG environment variable are:
  libs        display library search paths
  reloc       display relocation processing
  files       display progress for input file
  symbols     display symbol table processing
  bindings    display information about symbol binding
  versions    display version dependencies
  scopes      display scope information
  all         all previous options combined
  statistics  display relocation statistics
  unused      determined unused DSOs
  help        display this help message and exit
To direct the debugging output into a file instead of standard output
a filename can be specified using the LD_DEBUG_OUTPUT environment variable.
$
```

You can also show how symbols are resolved upon program execution, and help find things like linking to the wrong version of a library. Try:

```
$ LD_DEBUG=all  ls
```

to see a sample.