APPENDIX A
Running make

GNU make has an impressive set of command-line options. Most command-line options include a short form and a long form. Short commands are indicated with a single dash followed by a single character, while long options begin with a double dash usually followed by whole words separated by dashes. The syntax of these commands is:

```
-o argument
--option-word=argument
```

The following are the most commonly used options to make. For a complete listing, see the GNU make manual or type `make --help`.

`--always-make`
`-B`
: Assume every target is out of date and update them all.

`--directory=`*`directory`*
`-C` *`directory`*
: Change to the given directory before searching for a *makefile* or performing any work. This also sets the variable `CURDIR` to *`directory`*.

`--environment-overrides`
`-e`
: Prefer environment variables to *makefile* variables when there is a choice. This command-line option can be overridden in the *makefile* for particular variables with the `override` directive.

`--file=`*`makefile`*
`-f` *`makefile`*
: Read the given file as the *makefile* rather than any of the default names (i.e., *makefile*, *Makefile*, or *GNUMakefile*).

`--help`
`-h`
: Print a brief summary of the command-line options.

`--include-dir=`*`directory`*
`-I` *`directory`*
: If an include file does not exist in the current directory, look in the indicated directories for include files before searching the compiled-in search path. Any number of `--include-dir` options can be given on the command line.

`--keep-going`
`-k`
: Do not terminate the `make` process if a command returns an error status. Instead, skip the remainder of the current target, and continue on with other targets.

`--just-print`
`-n`
: Display the set of commands that would be executed by `make`, but do not execute any commands from command scripts. This is very useful when you want to know what `make` will do before actually doing it. Be aware that this option does not prevent code in `shell` functions from executing, just commands in command scripts.

`--old-file=`*`file`*
`-o` *`file`*
: Treat *`file`* as if it were infinitely old, and perform the appropriate actions to update the goals. This can be very useful if a file has been accidentally touched or to determine the effect of one prerequisite on the dependency graph. This is the complement of `--new-file` (`-W`).

`--print-data-base`
`-p`
: Print `make`'s internal database.

`--touch`
`-t`
: Execute the `touch` program on each out-of-date target to update its timestamp. This can be useful in bringing the files in a dependency graph up to date. For instance, editing a comment in a central header file may cause `make` to unnecessarily recompile an immense amount of code. Instead of performing the compile and wasting machine cycles, you can use the `--touch` option to force all files to be up to date.

`--new-file=`*`file`*
`-W` *`file`*
: Assume *`file`* is newer than any target. This can be useful in forcing an update on targets without having to edit or touch a file. This is the complement of `--old-file`.

`--warn-undefined-variables`

Print a warning message if an undefined variable is expanded. This is a useful diagnostic tool since undefined variables quietly collapse into nothing. However, it is also common to include empty variables in *makefile*s for customization purposes. Any unset customization variables will be reported by this option as well.