Makefiles - Recursive Make For Sub-directories

Large software projects generally are broken into several sub-directories, where each directory contains code that contributes to the whole.

The way it can be done is to do a recursive *make* descending into each sub-directory. To keep a common set of macros that are easily maintained we use the `include` statement which is fairly common in most *make*s

The following is the directory structure of the sources:

```
-rw-r-----   1 rk  owen     625 Jun 17 16:42 Makefile
-rw-r-----   1 rk  owen     142 Jun 17 16:43 Makefile.inc
-rw-r-----   1 rk  owen     133 Jun 17 14:32 main.c
-rw-r-----   1 rk  owen     120 Jun 17 14:34 proj.h

drwxr-x---   2 rk  owen    1024 Jun 17 16:54 subdira
-rw-r-----   1 rk  owen      45 Jun 17 14:28 subdira/sub2a.c
-rw-r-----   1 rk  owen      45 Jun 17 14:28 subdira/sub3a.c
-rw-r-----   1 rk  owen     346 Jun 17 16:34 subdira/Makefile
-rw-r-----   1 rk  owen      45 Jun 17 14:25 subdira/sub1a.c

drwxr-x---   3 rk  owen    1024 Jun 17 16:54 subdir
-rw-r-----   1 rk  owen     524 Jun 17 16:07 subdir/Makefile
-rw-r-----   1 rk  owen      52 Jun 17 14:33 subdir/sub1.c
-rw-r-----   1 rk  owen      52 Jun 17 14:33 subdir/sub2.c
-rw-r-----   1 rk  owen      52 Jun 17 14:33 subdir/sub3.c

drwxr-x---   2 rk  owen    1024 Jun 17 16:54 subdir/subsubdir
-rw-r-----   1 rk  owen      47 Jun 17 14:28 subdir/subsubdir/subsub3.c
-rw-r-----   1 rk  owen     390 Jun 17 16:35 subdir/subsubdir/Makefile
-rw-r-----   1 rk  owen      47 Jun 17 16:53 subdir/subsubdir/subsub1.c
-rw-r-----   1 rk  owen      47 Jun 17 14:28 subdir/subsubdir/subsub2.c
```

Here is the `Makefile.inc` and `Makefile` in the root:

`Makefile.inc`

```
# put common definitions in here
CC      = gcc
PRJCFLAGS       = -g
LD      = gcc
LDFLAGS =
AR      = ar
ARFLAGS =
RANLIB  = ranlib
RM      = rm
ECHO    = echo

SHELL   = /bin/sh

.SILENT :
```

Makefile

```
include Makefile.inc

DIRS    = subdir subdira
EXE     = mainx
OBJS    = main.o
OBJLIBS = libsub.a libsuba.a libsubsub.a
LIBS    = -L. -lsub -lsuba -lsubsub

all : $(EXE)

$(EXE) : main.o $(OBJLIBS)
        $(ECHO) $(LD) -o $(EXE) $(OBJS) $(LIBS)
        $(LD) -o $(EXE) $(OBJS) $(LIBS)

libsub.a libsubsub.a : force_look
        $(ECHO) looking into subdir : $(MAKE) $(MFLAGS)
        cd subdir; $(MAKE) $(MFLAGS)

libsuba.a : force_look
        $(ECHO) looking into subdira : $(MAKE) $(MFLAGS)
        cd subdira; $(MAKE) $(MFLAGS)

clean :
        $(ECHO) cleaning up in .
        -$(RM) -f $(EXE) $(OBJS) $(OBJLIBS)
        -for d in $(DIRS); do (cd $$d; $(MAKE) clean ); done

force_look :
        true
```

Which produces this output:

```
% make
looking into subdir : make -s
ar rv ../libsub.a sub1.o sub2.o sub3.o
a - sub1.o
a - sub2.o
a - sub3.o
ranlib ../libsub.a
looking into subsubdir : make -s
ar rv ../../libsubsub.a subsub1.o subsub2.o subsub3.o
a - subsub1.o
a - subsub2.o
a - subsub3.o
ranlib ../../libsubsub.a
looking into subdira : make -s
ar rv ../libsuba.a sub1a.o sub2a.o sub3a.o
a - sub1a.o
a - sub2a.o
a - sub3a.o
ranlib ../libsuba.a
looking into subdir : make -s
looking into subsubdir : make -s
gcc -o mainx main.o -L. -lsub -lsuba -lsubsub
```

Suppose we *touch* a file deep in the directory structure:

```
% touch subdir/subsubdir/subsub2.c
% make
looking into subdir : make -s
looking into subsubdir : make -s
ar rv ../../libsubsub.a subsub2.o
r - subsub2.o
ranlib ../../libsubsub.a
looking into subdira : make -s
looking into subdir : make -s
looking into subsubdir : make -s
gcc -o mainx main.o -L. -lsub -lsuba -lsubsub
```

Notice that the `Makefile` has a dummy target named `force_look` that the libraries depend on. This ``*file*'' is never created hence *make* will always execute that *target* and all that depend on it. If this was not done then *make* would have no idea that `libsubsub.a` depends on `subdir/subdir/subsub2.c` unless we include these dependencies in the root `Makefile`. This defeats the purpose of breaking up a project into separate directories. This mechanism pushes the dependency checking into lower level `Makefile`s.

Here is a representive sub-directory `Makefile` :

```
include ../Makefile.inc

CFLAGS  = $(PRJCFLAGS) -I..
OBJLIBS = ../libsub.a ../libsubsub.a
OBJS    = sub1.o sub2.o sub3.o

all : $(OBJLIBS)

../libsub.a : $(OBJS)
	$(ECHO) $(AR) $(ARFLAGS) rv ../libsub.a $?
	$(AR) $(ARFLAGS) rv ../libsub.a $?
	$(ECHO) $(RANLIB) ../libsub.a
	$(RANLIB) ../libsub.a

../libsubsub.a : force_look
	$(ECHO) looking into subsubdir : $(MAKE) $(MFLAGS)
	cd subsubdir; $(MAKE) $(MFLAGS)

clean :
	$(ECHO) cleaning up in subdir
	-$(RM) -f $(OBJS)
	cd subsubdir; $(MAKE) $(MFLAGS) clean

force_look :
	true
```

We use the pre-defined implicit rules, and define `CFLAGS` with project wide options and where to find the include files.

Notice the ganged shell commands to *cd* into a subdirectory and to execute a *make*. It's not for compactness or convenience ... it's required for correct behavior. The following section will detail some of these issues.

Slide 8