Tips & Tricks for GNU Make
(and Clearmake)

GNU Make is an extremely powerful build control tool--when used to its full potential.

- Paul's Rules of Makefiles
- Clearmake Workarounds
- Example Build Environment

Tips & Tricks for GNU Make

1st Rule of Makefiles:

Use GNU Make

GNU Make has many advantages over other makes:

- Cross-Platform: runs on any UNIX system. There are also ports to NT and DOS.
- Flexibility: combines the best features of most other makes, plus many unique features, into one package.
- Reliability: you get the sources, plus the entire Net and the FSF for bug fixes.

2nd Rule of Makefiles

Every non-.PHONY rule **must** update a file with the **exact** name of its target.

Many people try to create DO's in other directories with rules like this:

```
% : %.o                      or   foo.o: foo.c
        $(LINK) -o $(D)/$@             $(CC) -o $(D)/foo.o
```

This won't work! If you tell make that running a command will produce "foo.o", then it had better actually produce "foo.o", and not "../obj/foo.o". Don't lie to make; it will get its revenge.

Alternatives to the above rules might be:

```
$(D)/% : %.o                 or   $(D)/foo.o: foo.c
        $(LINK) -o $@                  $(CC) -o $@
```

These are better, but can get dreary to handle. See the next rule.

psmith@Baynetworks.com

3rd Rule of Makefiles

Life is simplest if targets are built in the current working directory.

Use VPATH to locate the sources from the objects directory, not to locate the objects from the sources directory.

If you put DO's in other directories, you will constantly have to remember to prefix filenames in your Makefiles with $(OBJ) (because of the Second Rule of Makefiles). This makes your Makefiles difficult to maintain and messy-looking.

However, VPATH is great for finding source dependencies; take advantage of that. If make is invoked from the $(OBJ) directory with VPATH pointing to the $(SRC) directory, all filenames can be written with no pathname prefixes.

See the example for a way to make this easy for users.

psmith@Baynetworks.com

4th Rule of Makefiles

Follow the Principle of Least Repetition.

Never write a filename more than once. This can be accomplished through a combination of make variables, pattern rules, automatic variables, and GNU make functions.

All automatic variables ($@, $*, $<, $^, etc.) are available for all commands in GNU make (some makes only provide some automatic variables for implicit (suffix) rules).

- VPATH will only work if you use automatic variables.
- Saves typing and typos.
- Reminds you to follow the 2nd Rule of Makefiles (as long as you don't prefix them with paths).

Clearmake Workarounds

A discussion of current problems I've encountered with clearmake -C gnu is available via FTP:

```
ftp://ftp.wellfleet.com/netman/psmith/atria/clearmake.archive
```

This is a number of archived email messages discussing clearmake in GNU make emulation mode.

Example Build Environment

The build environment we've developed at Bay Networks has these features:

- Multiple architectures created from the same source code.
- Different versions (debug, optimized, etc.) of the same architecture from the same source code.
- All DOs in a single directory for easy cleanup
- No sources in the DO directories
- Extensive use of VPATH to locate files

Directory Structure

- A tree consists of a hierarchy of directories. Each directory is an intermediate directory or a source directory.
- Intermediate directories contain a Makefile and subdirectories. The Makefile recursively makes all its subdirectories.
- Source directories contain a Makefile and source files. During builds, DO subdirectories are created and DOs are stored in them.
- DO subdirectories have the form `_<arch>_<osVersion>` or `_<alt>_<arch>_<osVersion>`

```
tree/ ---- Makefile
           hello/ ---- Makefile
                       hello.c
                       _sun4_4.1/ ---- hello.o
                                       hello
```

Items in bold are view-local or derived objects; other items are ClearCase elements.

Invoking Clearmake

We use a script “cmake” to invoke clearmake.

- Sets a very restricted PATH.
- Sets MAKEINCPATH to directories to search for included makefiles.
- Strips the environment with env(1).
- Invokes clearmake -C gnu.

```
echo "Build commencing on `date`"
echo "------------------------------------------------------------------------------"

CLEARMAKE=${CLEARMAKE:-"clearmake -C gnu"}
PATH=/tree/bin/$ARCH:/tree/bin:/bin:/usr/bin:${ATRIAHOME:-/usr/atria/bin}
MAKEINCPATH="-I /tree/mkfiles"

env - \
   PATH=$PATH TZ=$TZ LANG=$LANG LOGNAME=$LOGNAME TMPDIR=$TMPDIR \
   LD_LIBRARY_PATH=/tree/lib \
   CLEARCASE_BLD_UMASK=${CLEARCASE_BLD_UMASK:-2} \
   CLEARCASE_ROOT=$CLEARCASE_ROOT \
   MAKEINCPATH="$MAKEINCPATH" \
   $CLEARMAKE $MAKEINCPATH "$@"

echo "------------------------------------------------------------------------------"
echo "Build completed on `date`"
```


Global Makefiles

Global makefiles are included by regular makefiles to define common rules, variables, etc. There may be some overlap here with BOS files?

- Makefile_Init defines variables needed by all invocations of make
- Makefile_Common defines common variables for source makefiles, but no rules
- Makefile_Rules defines all common pattern rules for source makefiles, and also contains VPATH/vpath statements
- Makefile_Subdirs defines rules for recursively invoking make from intermediate directories
- Makefile_Arch defines rules for re-invoking make in DO subdirectories

Intermediate Directory Makefiles

Makefiles for intermediate directories have the following format:

- Include Makefile_Init
- Define the variable SUBDIRS to contain all the subdirectories to be built
- Define any order-dependencies
- Include Makefile_Subdirs

```
include Makefile_Init
#-----------------------------------------

SUBDIRS  = hellolib hello

hello: hellolib

#-----------------------------------------
include Makefile_Subdirs
```


Contents of Makefile_Init

Makefile_Init contains just a couple of variable definitions which are needed by almost every invocation of make:

```
ARCH    := $(shell warch)
TREE    := /tree
PWD     := $(shell pwd | sed s@/tmp_mnt@@)

install::
```

- The “warch” shell script uses uname(1) to determine the host architecture and prints a canonical version as described earlier
- The install rule always appears as the first rule in all makefiles, since all makefiles include Makefile_Init first, so unless another target is given on the cmake command line the “include” rule is always run.

Contents of Makefile_Subdirs

Makefile_Subdirs defines rules for recursively invoking submakes from intermediate directory makefiles:

- Declare the subdirectories to be .PHONY so they'll always be "rebuilt".
- The "install" rule is defined to depend on the subdirectories
- "cmake *subdirectory*" explicitly recursively makes that directory
- "cmake *rule*" recursively makes *rule* in each subdirectory

```
.PHONY: $(SUBDIRS)
MAKE.dir = cd $@; $(MAKE) $(MAKEINCPATH)
MAKE.subdirs = @for dir in $(SUBDIRS); do \
                 (cd $$dir; exec $(MAKE) $(MAKEINCPATH) $@); done

install:: $(SUBDIRS)

$(SUBDIRS):
	+$(MAKE.dir)

.DEFAULT:
	+$(MAKE.subdirs)
```


Source Directory Makefiles

Source directory makefiles have the following format:

- Include Makefile_Init
- Check to see if we're in the DO subdirectory
- If not, include Makefile_Arch
- If so, include Makefile_Common, define the rules to build the source directory, then include Makefile_Rules last.

```
include Makefile_Init
ifneq ($(notdir $(PWD)),$(ARCH))
include Makefile_Arch
else
include Makefile_Common
#---------------------------------------------

   [...normal makefile...]

#---------------------------------------------
include Makefile_Rules
endif
```

Contents of Makefile_Arch

Makefile_Arch contains the rules to build the DO subdirectory and re-invoke make in that subdirectory. This keeps the developer from having to cd to the DO subdirectory to build the product:

- Remove all default rules
- Define all rules to run the recursive make

```
.SUFFIXES:
MAKE.arch = @if [ ! -d $(ARCH) ]; then \
               mkdir $(ARCH); fi; \
            cd $(ARCH); \
            $(MAKE) $(MAKEINCPATH) -f ../Makefile

.DEFAULT all:
        +$(MAKE.arch) $@

install::
        +$(MAKE.arch) $@
```


Contents of Makefile_Common

Makefile_Common contains all the common make variables for the project, very much like a BOS file. In addition to normal make variables, its most important function is to define VPATH variables:

- VPATH.s is the source directory: searched first
- VPATH.h is for header files
- VPATH.a is for libraries
- Include an architecture-specific Makefile at the bottom to override commands for specific architectures

```
PWD_UP   := $(patsubst %/,%,$(dir $(PWD)))
VPATH.s  = $(PWD_UP)
VPATH.h  = $(TREE)/include $(X11R5)/include
VPATH.a  = $(TREE)/lib $(X11R5)/lib $(SYS)/lib

include Makefile$(ARCH)
```

Contents of Makefile_Rules

Makefile_Rules defines the pattern rules for common build scripts in source directories. Its other major function is to define variables using VPATH and vpath rules:

```
vpath % $(VPATH.s)
vpath %.h $(VPATH.h)
vpath %.a $(VPATH.a)
vpath % $(VPATH.g)

CPPPATH  := -I. $(VPATH.h:%=-I%)
LDPATH   := $(VPATH.a:%=-L%)

install:: install_hdrs all install_bins install_libs install_alt

clean::
	$(RM) -f *

remake: clean install

.PHONY: all install clean remake \
        install_bins install_libs install_hdrs install_alt
```