Make and Makefiles

Last Edit
March 23, 1992

Reg Quinton <reggers@julian.uwo.ca>
Computing and Communications Services
The University of Western Ontario
London, Ontario N6A 5B7
Canada

1. History/Goals

Make is one of the original Unix tools for Software Engineering. By S.I. Feldman of AT&T Bell Labs circa 1975. But there are public domain versions (eg. GNU) and versions for other systems (eg. Vax/VMS).

Related tools are the language compilers (cc, f77, lex, yacc, etc.) and shell programming tools (eg. awk, sed, cp, rm, etc.). You need to know how to use these.

Important adjuncts are lint (source code checking for obvious errors) ctags (locate functions, etc. in source code) and mkdepend. These are nice, and good programmers use them.

Important, and related tools, are the software revision systems SCCS (Source Code Control System -- this is a licenced product) and RCS (Revision Control System -- this is public domain and the recommended choice of CCS). You *ought* to use these.

The idea is to automate and optimize the construction of programs/files — ie. to leave enough *foot prints* so that others can follow.

1.1. Where to start?

Like anything else (apart from the `cat` and `ls` commands) `make` can be frightening the first time you encounter it — but it needn't be. You can learn about `make` by:

(1) Attend a course, like this one! Congratulations!

(2) RTFM (read the f'ing manual). Available on **all** Unix systems.

(3) Read more manuals: *Make — A Program for Maintaining Computer Programs*, by S.I. Feldman (in) BSD and System V manauls.

(4) Follow the examples set in comp.unix.sources, comp.misc.sources, /usr/src, etc. **Beware:** the examples set in alt.sources are pretty bad — tools are often distributed without decent Makefiles.

(5) Start small and build on what you've done.

(6) Start today, don't wait until later. It's easy once you get started.

Compare with figuring out your VCR, digital watch, camera, etc. This takes a bit of time but isn't too bad once you get going.

Figuring out how to use Makefiles will take some time, but it's time well spent.

1.2. An example

These slides are produced by Make!

```
[1:11pm suncon] ls
Makefile  RCS       README    sizing    slides    title

[1:11pm suncon] page README
This is the slides for my Makefile course. Type "make" t
produce a postscript version suitable for printing on a
laser printer.

        Reg Quinton; 30-Oct-90

[1:11pm suncon] make
groff -me -Tps sizing title slides >paper.ps
```

The goal is to be able to approach any software distribution and just type

```
[11:32am shadow] make
```

and/or

```
[11:32am suncon] make all {or install, clean, depend...}
```


1.3. Makefiles

The rules for constructing targets are itemized in a **Makefile**

```
[11:26am suncon] ls
Makefile  RCS       README    sizing    slides    title
```

All *good* software distributions will have a Makefile and README to get you started. If you're going to install something, it better have a manual as well.

Any project you expect others to be involved with (eg. when you're away who fixes your problems?) better have both. *You have to leave foot prints*.

The ideal README will tell you to do a few minor things (like change a macro definition for BIN, MAN, or etc) and then say "make".

Any ongoing project should have be also managed with RCS (more on that another day) or SCCS.

1.4. Makefile Naming

Make is going to look for a file called **Makefile**, if not found then a file called **makefile**. Use the first (so the name stands out in listings).

You can get away without any Makefile (but shouldn't)! Make has default rules it knows about.

```
[12:01pm julian] ls
printenv.c

[12:01pm julian] page printenv.c
#include        <stdio.h>
main(argc,argv,argp)
int     argc;
char    *argv[],*argp[];
{
  char  **ep;
  for (ep=argp; *ep; ep++) printf("%s\n",*ep);
  exit(0);
}
```

Do I have to resort to figuring out and using the C compiler (loader, assembler, etc) to build the executable?

Even if you don't have a Makefile you can still use the make command:

```
[12:01pm julian] make printenv
        cc  printenv.c  -o printen

[12:01pm julian] ./printenv
EDITOR=emacs
HOME=/usr/people/guru/reggers
HOST=julian.uwo.ca
NNTPSERVER=newshost.uwo.ca
ORGANIZATION=University of Western Ontario, London
etc...
```

Using a make command is much easier than figuring out how to run a compiler and fill in the right switches, etc. That can be delayed until later.

In any case, make seems to know a lot about the C compiler that would be useful for the first time programmer.

1.5. Makefile Syntax/Semantics

```
#
# $Author: reggers $
# $Date: 1992/02/14 21:00:58 $
# $Id: slides,v 1.2 1992/02/14 21:00:58 reggers Exp $
#

SRCS=    sizing title slides
DOC=     /usr/doc/make.course.ps
MACROS=  -me
PSROFF=  groff -Tps
DITROFF= groff -Tdvi

paper.ps: $(SRCS)
        $(PSROFF) $(MACROS) $(SRCS) >paper.ps

paper.dvi: $(SRCS)
        $(DITROFF) $(MACROS) $(SRCS) >paper.dvi

preview: paper.dvi
        xdvi paper.dvi

install: paper.ps
        psnup -p2 paper.ps >$(DOC)
        chmod a+r $(DOC)

clean:
        -rm core *~ paper.ps paper.dvi
```


This shouldn't look like magic. There's some comments, some macros defined (or variables), some dependency rules (make the paper if the sources have changed), and some commands to make things.

1.6. Makefile components

Comments are any text beginning with the pound (#) sign:

```
# $Id: slides,v 1.2 1992/02/14 21:00:58 reggers Exp $
```

Macros. Make has a simple macro definition and substitution mechanism:

```
MACROS=  -me
PSROFF=  groff -Tps
DITROFF= groff -Tdvi
```

Targets. You make a particular target (eg. **make all**), in none specified then the first target found:

```
paper.dvi: $(SRCS)
        $(DITROFF) $(MACROS) $(SRCS) >paper.dvi
```

The target is made if any of the dependent files have changed.

Continuation lines. Use a back slash (\). This is important for long macros and/or rules — more on this in a moment.

1.7. Makefile Macros

Macros are defined in a Makefile as **<string>=<value>**; for example:

```
CFLAGS= -O -systype bsd43
```

There are lots of default macros — you should honor the existing naming conventions.

To find out what rules/macros make is using type:

```
[1:23pm suncon] make -p
etc...
USER= reggers
TERM= xterm
etc...
LD= ld
AS= as
AR= ar
FC= f77
CC= cc
etc...
```

Note that your environment variables are exported into the make as macros. They will override the defaults. For example:

```
[2:38pm julian] ls
printenv.c

[2:39pm julian] setenv CFLAGS -O
[2:39pm julian] setenv LDFLAGS -s
[2:41pm julian] make printenv
        cc -O printenv.c -s -o printenv
```

Compare with our earlier example:

```
[12:01pm julian] make printenv
        cc  printenv.c  -o printenv
```

Alternatively, you can set macros on the make command line:

```
[3:40pm julian] make "CFLAGS= -O" "LDFLAGS=-s" printenv
        cc -O printenv.c -s -o printenv
```

Both of these are strategies you (generally) ought to avoid. But they can be handy.

A good example of importing a macro from the command line

```
[3:40pm julian] make "PSROFF=nroff"
```

This would make an nroff (ie. plain text) version of my slides.

Another good example of importing a macro from the command line

```
[3:40pm julian] su
Password:

Root: make "OWNER=reggers" install
install -c -m 755 -o reggers printenv /usr/local/bin
```

etc...

This strategy is a good method for making sure that programs are installed with the ownership tagged to a particular person.

Most install targets use the OWNER macro. But, browse the Makefile first to see how the install is done. Alternatively use **make -n**:

```
Root: make -n "OWNER=reggers" install
install -c -m 755 -o reggers printenv /usr/local/bin
```
etc...

The **-n** flag says: tell me what you would do (but don't do it). Ie. it displays the commands that would be executed.

You **cannot** safely assume that everyone has the same environment variables that you have. Therefore it is best to construct a Makefile and define your macros — so that others can build things too.

```
[2:53pm julian] ls
Makefile    printenv.c

[2:53pm julian] page Makefile
# make the printenv command

CFLAGS= -O
LDFLAGS= -s
CC=cc

printenv: printenv.c
        $(CC) $(CFLAGS) printenv.c \
              $(LDFLAGS) -o printenv

clean:
        -rm printenv *~ *.o core

[2:53pm julian] make
        cc -O printenv.c -s -o printenv
```

This Makefile constructed from the make defaults learned from make.

1.8. Conventional Macros

There are lots of default macros (type "make -p" to print out the defaults). Most are pretty obvious from the rules in which they are used:

```
AR = ar
GFLAGS =
GET = get
ASFLAGS =
MAS = mas
AS = as
FC = f77
CFLAGS =
CC = cc
LDFLAGS =
LD = ld
LFLAGS =
LEX = lex
YFLAGS =
YACC = yacc
LOADLIBS =
MAKE = make
MAKEARGS = 'SHELL=/bin/sh'
SHELL = /bin/sh
MAKEFLAGS = b
```


1.9. Special Macros

Before issuing any command in a target rule set there are certain special macros predefined.

(1) **$@** is the name of the file to be made.

(2) **$?** is the names of the changed dependents.

So, for example, we could use a rule

```
printenv: printenv.c
        $(CC) $(CFLAGS) $? $(LDFLAGS) -o $@
```

alternatively:

```
printenv: printenv.c
        $(CC) $(CFLAGS) $@.c $(LDFLAGS) -o $@
```

There are two more special macros used in *implicit rules* (which we will discuss in a moment). They are:

(1) **$<** the name of the related file that caused the action.

(2) **$*** the prefix shared by target and dependent files.

2. Makefile Target Rules

The general syntax of a Makefile Target Rule is

target [*target*...] : [*dependent* ....]
[<tab> *command* ...]

Items in brackets are optional, ellipsis means one or more. Note the **tab** to preface each command is required (watch out emacs users!).

The semantics is pretty simple. When you say "make target" make **finds** the target rule that applies and, if any of the dependents are **newer** than the target, make **executes** the commands one at a time (after macro substitution). If any dependents have to be made, that happens first (so you have a recursion).

A make will **terminate** if any command returns a failure status. That's why you see rules like:

```
clean:
        -rm *.o *~ core paper
```

Make **ignores** the returned status on command lines that begin with a dash. eg. who cares if there is no core file?

Make will **echo** the commands, after macro substition to show you what's happening as it happens. Sometimes you might want to turn that off. For example:

```
install:
        @echo You must be root to install
```

There's also a method for turning off all the echos.

.SILENT:

The pseudo target, .SILENT, if found will defeat the echoing. This is a **bad** idea and should be used with care.

Never trust a software distribution with the .SILENT target — someone's up to no good if they're using it. Trojan horse perhaps?

2.1. Example Target Rules

For example, to manage sources stored within RCS (sometimes you'll need to "check out" a source file):

```
SRCS=x.c y.c z.c

$(SRCS):
        co $@
```

To manage sources stored within SCCS (sometimes you'll need to "get" a source file):

```
$(SRCS):
        sccs get $@
```

Alternativley, to manage sources stored within SCCS or RCS let's generalize with a macro that we can set as required.

```
SRCS=x.c y.c z.c
# GET= sccs get
GET= co

$(SRCS):
        $(GET) $@
```


For example, to construct a library of object files

```
lib.a: x.o y.o z.o
        ar rvu lib.a x.o y.o z.o
        ranlib lib.a
```

Alternatively, to be a bit more fancy you could use:

```
OBJ=x.o y.o z.o
AR=ar

lib.a: $(OBJ)
        $(AR) rvu $@ $(OBJ)
        ranlib $@
```

Since AR is a default macro already assigned to "ar" you can get away without defining it (but shouldn't).

If you get used to using macros you'll be able to make a few rules that you can use over and over again.

For example, to construct a library in some other directory

```
INC=../misc
OTHERS=../misc/lib.a

$(OTHERS):
        cd $(INC); make lib.a
```

Beware:, the following will not work (but you'd think it should)

```
INC=../misc
OTHERS=../misc/lib.a

$(OTHERS):
        cd $(INC)
        make lib.a
```

Each command in the target rule is executed in a separate shell. This makes for some interesting constructs and long continuation lines. For example,

```
install:  $(libs) $(special) $(public) $(private) nntp_s
        for i in $(public); do \
                $(install) -c -m 755 $$i $(rnbin)/$i; \
        done
```


To generate a tags file

```
SRCS=x.c y.c z.c
CTAGS=ctags -x >tags

tags:   $(SRCS)
        ${CTAGS} $(SRCS)
```

On large projects a tags file, that lists all functions and their invocations is a handy tool. (But I always use grep).

To generate a listing of likely bugs in your problems

```
lint:
        lint $(CFLAGS) $(SRCS)
```

Henry Spencer's first rule of programming is always use lint. Lint is a really good tool for finding those obvious bugs that slip into programs — eg. type classes, bad argument list, etc. (But I've never used it).

2.2. All:, Install:, Clean:,..

People have come to expect certain targets in Makefiles. You should always browse first, but it's reasonable to expect that the targets **all** (or just make), **install**, and **clean** will be found.

(1) **make all** — should compile everything so that you can do local testing before installing things.

(2) **make install** — should install things in the right places. **But** watch out that things are installed in the right place for *your system*.

(3) **make clean** — should clean things up. Get rid of the executables, any temporary files, object files, etc.

You may encounter other common targets, some have been already mentioned (tags and lint).

3. printenv Makefile

```
# make the printenv command, Reg Quinton.
#
# $Author: reggers $
# $Date: 1992/02/14 21:00:58 $
# $Id: slides,v 1.2 1992/02/14 21:00:58 reggers Exp $

OWNER=bin
GROUP=bin

CTAGS= ctags -x >tags
CFLAGS= -O
LDFLAGS= -s
CC=cc
GET=co
SRCS=printenv.c
OBJS=printenv.o
SHAR=shar
MANDIR=/usr/man/manl/printenv.l
BINDIR=/usr/local/bin

# On mips (as lifted from BSD)
#DEPEND= mkdepend -c "$(CC) $(CFLAGS)" Makefile
# Or the X11 version (again, as lifted from BSD)
DEPEND= makedepend $(CFLAGS)
# Hmmm... have never used mkmkf
#DEPEND = mkmkf
```

```
all:    printenv

# To get things out of the revision control system

$(SRCS):
        $(GET) $@

# To make an object from source

        $(CC) $(CFLAGS) -c $*.c

# To make an executable

printenv: $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS)

# To install things in the right place

install: printenv printenv.man
        $(INSTALL) -c -o $(OWNER) -g $(GROUP) -m 755 printenv $(E
        $(INSTALL) -c -o $(OWNER) -g $(GROUP) -m 644 printenv.mar

# where are functions/procedures?

tags: $(SRCS)
        $(CTAGS) $(SRCS)
```



```
# what have I done wrong?

lint: $(SRCS)
        lint $(CFLAGS) $(SRCS)

# what are the source dependencies

depend: $(SRCS)
        $(DEPEND) $(SRCS)

# to make a shar distribution

shar:   clean
        $(SHAR) README Makefile printenv.man $(SRCS) >shar

# clean out the dross

clean:
        -rm printenv *~ *.o *.bak core tags shar

# DO NOT DELETE THIS LINE -- make depend depends on it.

printenv.o: /usr/include/stdio.h
```

3.1. Mlnet Makefile

```
# $Author: reggers $
# $Locker:  $
# $Date: 1992/02/14 21:00:58 $
# $Id: slides,v 1.2 1992/02/14 21:00:58 reggers Exp $
# $Source: /usr/src/usr.local/doc/courses/make/RCS/slides,v $
# $State: Exp $
#
# Title:  make
#
# Date:   Feb 23, 1984
#
# Author: Reginald E. Quinton
#
# Abstract:
#       dependencies for mlcico.
#
#       what is the daemon user and group?
DAEMON=uucp
GROUP=daemon
PATH=/usr/bsd43/bin:/usr/ucb:/usr/bin:/usr/local:/bin:/usr/etc:/e
#       where are executables put?
BIN=/usr/ccs/bin/mlnet
#       where do we save old junk
OLD=/usr/ccs/old
#       where are the spool and lib areas?
SPLIB=/usr/lib/mlnet /usr/spool/mlnet $(BIN)
```



```
#       what do I use as lex?
LEX=lex
#       BSD style install
INSTALL=install
#       compilation flags
# The following on sun3
#LDFLAGS= -lresolv
#CFLAGS= -c -O -DTCPIP -DPIPE -DMLCICO=\"$(BIN)/mlcico\" \
#       -DTERMIO=\<sys/termio.h\> \
#       -DLOG_MLNET=LOG_LOCAL1 -DMAXDGL=200 -DMAXPPI=4 -I../misc
# The following, on mips
LDFLAGS=
CFLAGS= -c -O -DTCPIP -DPIPE -DMLCICO=\"$(BIN)/mlcico\" \
        -DTERMIO=\<sysv/sys/termio.h\> -systype bsd43 \
        -DLOG_MLNET=LOG_LOCAL1 -DMAXDGL=200 -DMAXPPI=4 -I../misc
#       -DDECNET -DTCPIP -DPIPE would be nice too.
#       there are some other externals, where are they?
INC=../misc
OTHERS=../misc/lib.a

# Make all assumes a certain environment

all:    lib.a $(OTHERS) mkhost mlcico mlcall sendmlnet
```

```
# These force a particular environment

mips:
        make all "LDFLAGS= "\
        "CFLAGS= -c -O -DTERMIO=\<sysv/sys/termio.h\> -systype bs
        -DTCPIP -DPIPE -DMLCICO=\\\"$(BIN)/mlcico\\\" \
        -DLOG_MLNET=LOG_LOCAL1 -DMAXDGL=200 -DMAXPPI=4 -I../misc"

sun3:
        make all "LDFLAGS= -lresolv -Bdynamic" \
        "CFLAGS= -c -O -DTERMIO=\<sys/termio.h\> \
        -DTCPIP -DPIPE -DMLCICO=\\\"$(BIN)/mlcico\\\" \
        -DLOG_MLNET=LOG_LOCAL1 -DMAXDGL=200 -DMAXPPI=4 -I../misc"

$(OTHERS):
        cd $(INC); make lib.a

HFILES= glob.h types.h

SRCS=   akthis.c baud.c bld_dcb.c decnet.c device.c getenv.c inms
        interpret.c kermit.c match.c mkhost.l mlcall.c mlcico.c \
        nakthis.c nxmsg.c panic.c pipe.c process.c response.c rma
        sendmlnet.c stimulate.c tcpip.c glob.h types.h

$(SRCS):
        co $@

sources: $(SRCS)
```



```
OBJ=    akthis.o        baud.o          bld_dcb.o       decnet.o\
        device.o        getenv.o        inmsg.o\
        interpret.o     kermit.o        match.o\
        nakthis.o       nxmsg.o         process.o\
        response.o      rmail.o         rts.o\
        stimulate.o     tcpip.o         panic.o\
        pipe.o

        $(CC) $(CFLAGS) $*.c

lib.a: $(OBJ)
        ar rvu lib.a $(OBJ)
        ranlib lib.a

mkhost:  lib.a $(OTHERS) lex.yy.o
        $(CC) -s -o mkhost $(LDFLAGS) lex.yy.o lib.a $(OTHERS) -r
        @size mkhost

lex.yy.o: mkhost.l types.h
        $(LEX) mkhost.l
        $(CC) $(CFLAGS) -DMKHOST lex.yy.c

mlcico:  mlcico.o lib.a $(OTHERS)
        $(CC) -s -o mlcico $(LDFLAGS) mlcico.o lib.a $(OTHERS)
        @size mlcico

mlcall:  mlcall.o $(OTHERS)
        $(CC) -s -o mlcall $(LDFLAGS) mlcall.o $(OTHERS)
        @size mlcall
```

```
sendmlnet:    sendmlnet.o $(OTHERS)
        $(CC) -s -o sendmlnet $(LDFLAGS) sendmlnet.o $(OTHERS)
        @size sendmlnet

clean:
        -rm *.o *.old lib.a mlcico mkhost mlcall sendmlnet lex.yy

#    make the lib and spool areas owner DAEMON, group GROUP

$(SPLIB):
        mkdir $@
        chown $(DAEMON) $@
        chgrp $(GROUP) $@
        chmod 775 $@

#    $(INSTALL) mlcico, mkhost, make spool and lib if required

install: $(SPLIB)
        -mv $(BIN)/mkhost $(OLD)
        $(INSTALL) -c -g $(GROUP) -o $(DAEMON) -m 6750 mkhost $(B
        -mv $(BIN)/mlcico $(OLD)
        $(INSTALL) -c -g $(GROUP) -o $(DAEMON) -m 6755 mlcico $(B
        -mv $(BIN)mlcall $(OLD)
        $(INSTALL) -c -g $(GROUP) -o $(DAEMON) -m  755 mlcall $(B
        -mv $(BIN)/sendmlnet $(OLD)
        $(INSTALL) -c -g $(GROUP) -o $(DAEMON) -m  755 sendmlnet
        @echo "you need libs, spools, and filters for each host"

#    depend, the usual
```



```
depend:
        chmod 644 Makefile
        mkdep $(CFLAGS) $(SRCS)

# if you want to use /usr/local/lib/mlnet (ie. shared between mac

localib:
        mkdir -p /usr/local/lib/mlnet
        ln -s /usr/local/lib/mlnet /usr/lib/mlnet

# if you want to use /var/local/lib/mlnet (ie. private to each ma

varlib:
        mkdir -p /var/local/lib/mlnet
        ln -s /var/local/lib/mlnet /usr/lib/mlnet

# DO NOT DELETE THIS LINE -- mkdep uses it.
# DO NOT PUT ANYTHING AFTER THIS LINE, IT WILL GO AWAY.
```

3.2. Makefile Implicit Rules

Consider the rule we used for printenv

```
printenv: printenv.c
        $(CC) $(CFLAGS) printenv.c $(LDFLAGS) -o printen
```

We generalized a bit to get

```
printenv: printenv.c
        $(CC) $(CFLAGS) $@.c $(LDFLAGS) -o $@
```

The command is one that ought to work in all cases where we build an executable **x** out of the source code **x**.c This can be stated as an implicit rule:

```
.c:
        $(CC) $(CFLAGS) $@.c $(LDFLAGS) -o $@
```

This *Implicit* rule says how to make **x** out of **x**.c — run cc on **x**.c and call the output **x**. The rule is implicit because no particular target is mentioned. It can be used in all cases.

This implicit rule is built into make (type "make -p") and was responsible for building printenv when we had no Makefile.

```
[12:01pm julian] ls
printenv.c

[12:01pm julian] make -p
      etc...
.c:
        $(CC) $(CFLAGS) $@.c $(LDFLAGS) -o $@
      etc...

[12:01pm julian] make printenv
        cc  printenv.c  -o printen
```

An important observation here is:

- If you don't know how to build something (eg. and executable from source, a library of object, or etc.) then make either already knows how to do it, or can give you good clues on how it ought to be done.
- The paper by Feldman lists some of the implicit rules in the appendix — a good starting point.

Another common implicit rule is for the construction of .o (object) files out of .c (source files).

```
.o.c:
        $(CC) $(CFLAGS) -c $<
```

alternatively

```
.o.c:
        $(CC) $(CFLAGS) -c $*.c
```

You'll see lots of Makefiles that use this rule (it's usually stated but could be assumed).

```
        etc...
OBJS=x.o y.o z.o
        etc...
.c.o:
        $(CC) $(CFLAGS) -c $*.c
        etc...
program: $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS)
        etc...
```

The only problem (a very minor one) is you don't have anything listed in the dependencies (other than the defaults — that **x**.o depends on **x**.c for all values of **x**).

4. Make dependencies

It's pretty common to have source code that uses include files. For example:

```
[12:01pm julian] page program.c

#include        <stdio.h>
#include        "defs.h"
#include        "glob.h"
        etc... .
main(argc,argv)
        etc...
```

The implicit rule only covers part of the source code dependency (it only knows that program.o depends on program.c). The usual method for handling this is to list the dependencies separately;

```
        etc...
        $(CC) $(CFLAGS) -c $*.c
        etc...
program.o: program.c defs.h glob.h
```

Usually an implicit rule and a separate list of dependencies is all you need. And it ought to be easy enough to figure out what the dependencies are.

However, there are a number of nice tools around that will automatically generate dependency lists for you. For example (trivial)

```
DEPEND= makedepend $(CFLAGS)
        etc...
# what are the source dependencies

depend: $(SRCS)
        $(DEPEND) $(SRCS)

        etc....
# DO NOT DELETE THIS LINE -- ....

printenv.o: /usr/include/stdio.h
```

These tools (mkdepend, mkmkf, etc.) are very common these days and aren't too difficult to use or understand. They're just shell scripts that run cpp (or cc -M, or etc.) to find out what all the include dependencies are. They then just tack the dependency list onto the end of the Makefile.

For most projects this isn't required. If you're building something large and interdependent (eg. the kernel, or X11, or etc.) you'll find them essential.

These tools are usually public domain, rn comes with one (and acknowledges the BSD source).

5. Other uses of Makefiles

People often are of the impression that Makefiles are only useful for managing software projects — compile, install, etc. This is a **serious oversight**.

Make is a tool for managing file system dependencies (this file has to be rebuilt if one of these files have changed). Software management is only an instance of that. Make can be used creatively to manage much more.

```
[12:59pm julian] pwd
/usr/spool/cron/crontabs

[12:59pm julian] page root
        etc...
0       0       * * * \
        cd /usr/local/lib; make aliases >/dev/null
        etc...
0,15,30,45    * * * * \
        cd /usr/local/lib/named/data; make install >/dev
```

We have probably 10 or so different make jobs run on a regular basis out of crontab, or out of jobs run out of cron.

The Usenet group comp.mail.maps is used to distribute the maps required for the UUCP mailpath data base. We need to capture these maps if any have arrived. This is a pretty classic instance of a file system dependency.

```
#    Makefile: how to rebuild the uucp map data-base
#
#    Reg Quinton <reggers@hadrian.uwo.ca>; Aug 3, 1988
#
#    (cd /usr/spool/maps; make DATE) in crontab once each day
#

# directory where we store raw maps.
MAPS=/usr/spool/maps

# maps as received from USENET.
UMAPS=/usr/spool/news/comp/mail/maps/[0-9]*

# Notify who about work I do?

NOTIFY=reggers

DATE: ${UMAPS}
        for i in $?;  do \
         sed '1,/^$$/d' $$i | (cd ${MAPS}; /bin/sh); \
        done | mail -s "UUCP maps extracted" ${NOTIFY}
        touch ${MAPS}/DATE
```


We rebuild our sendmail aliases once a day (if required) using the name server data (ie. whois reggers), /etc/passwd, and other files.

```
# $Author: reggers $
# $Date: 1992/02/14 21:00:58 $
# $Id: slides,v 1.2 1992/02/14 21:00:58 reggers Exp $
#
#    Makefile: how to rebuild the alias file
#           on our server, and flood some clients
#
#    Reg Quinton <reggers@hadrian.uwo.ca>; July 26, 1988
#
#       19-Oct-90, req: automate Usenet mail interface
#
#    (cd /usr/ccs/lib; make aliases) in crontab once each day
#

aliases: aliases.public aliases.usenet aliases.private aliases.us
        cat aliases.public aliases.usenet \
         aliases.users aliases.private >/tmp/aliases
        cp /dev/null /tmp/aliases.dir
        cp /dev/null /tmp/aliases.pag
        /usr/lib/sendmail -bi -oA/tmp/aliases
        cp /tmp/aliases .
        cp /tmp/aliases.dir .
        cp /tmp/aliases.pag .
```

```
aliases.users: /etc/passwd
        awk -F: '{printf "%s: %s\n", $$1, $$1}' /etc/passwd | \
         sort >aliases.users

aliases.usenet: /usr/lib/news/active.times
        awk '{printf "%s: \"|/usr/lib/news/inews -h -n %s; exit (
         /usr/lib/news/active | sort >aliases.usenet

aliases.public: /usr/ccs/lib/directory/ALIASFILE
        cd /usr/ccs/lib/directory; make
        sed "s/\([^:]*\)\([^<]*\)<\([^>]*\).*/\1: \3/" \
         /usr/ccs/lib/directory/ALIASFILE >aliases.public
        /usr/ccs/lib/directory/ibmfold /usr/ccs/lib/directory/ALI
         mail -s update mdir@uwocc1.uwo.ca
        mail -s "Campus E-mail directory" campus@student.business
         < /usr/ccs/lib/directory/ALIASFILE
```


We automatically update the Domain Name Service every 15 minutes if any of the DNS data files have changed.

```
#    Makefile
#
# $Author: reggers $
# $Date: 1992/02/14 21:00:58 $
# $Locker:  $
# $Id: slides,v 1.2 1992/02/14 21:00:58 reggers Exp $
# $Source: /usr/src/usr.local/doc/courses/make/RCS/slides,v $
#
#    Hmmm... what are we up to? Well, I'd like to keep the
#    name->number and number->name mappings in the same file.
#    I construct the number->name mappings (in-addr.arpa.) out
#    of the first assuming a minimal responsible management of
#    the one file.

PATH=:/usr/bsd43/bin:/usr/ucb:/usr/bin:/usr/local:/bin:
PUBHOSTS=/usr/src/guru/ftp/nic/hosts
HOSTS=/etc/hosts
SOA=/usr/local/lib/lists/soa
POSTMAST=/usr/local/lib/lists/postmasters
BOOT=../named.boot
REVADDR=../in-addr.arpa
ETHRADDR=../in-ether.arpa
NOVLADDR=../in-novell.arpa
```

```
DOMAINS= apmaths.domain arts.domain astro.domain biostats.domain
        business.domain ccs.domain cogsci.domain edu.domain \
        engga.domain engrg.domain express.domain gfx.domain gp.do
        gsoj.domain hhcru.domain hs.ccs.domain library.domain net
        physics.domain physiology.domain psych.domain stats.domai
        slis.domain sscl.domain gfx.engga.domain dast.domain cls.
        scilab.ccs.domain housing.domain registrar.domain pt.doma
        fireprev.domain ohs.domain staffhlth.domain uwo.domain

all: install

$(DOMAINS):
        co $@

#
# This one is official, the in-addr.arpa table to map numbers bac
#
$(REVADDR): $(DOMAINS) in-addr.arpa bottom.in-addr.arpa
        @if [ ! -w $(REVADDR) ]; then \
         echo "You need more priviledges\n"; exit 1; \
        fi
        for i in $(DOMAINS); do\
         grep 100.129.in-addr.arpa $$i;\
        done | sed 's/^;//' | sort >.tmp-rev
        (cat in-addr.arpa | sed s/VERSION/`cat install`/;\
         echo '; #### CUT HERE ####' `date` '#### CUT HERE ####',
              cat .tmp-rev bottom.in-addr.arpa) > $(REVADDR)
        rm .tmp-rev
```



```
#
# This one is local, map ether numbers back to names as above
#
$(ETHRADDR): $(DOMAINS) in-ether.arpa
        @if [ ! -w $(ETHRADDR) ]; then \
         echo "You need more privileges\n"; exit 1; \
        fi
        -for i in $(DOMAINS); do\
         grep in-ether.arpa $$i;\
        done | sed 's/^;//' | sort >.tmp-ether
        (cat in-ether.arpa | sed s/VERSION/`cat install`/;\
         echo '; #### CUT HERE ####' `date` '#### CUT HERE ####',
              cat .tmp-ether) > $(ETHRADDR)
        rm .tmp-ether

$(NOVLADDR): $(DOMAINS) in-novell.arpa
        @if [ ! -w $(NOVLADDR) ]; then \
         echo "You need more privileges\n"; exit 1; \
        fi
        -for i in $(DOMAINS); do\
         grep in-novell.arpa $$i;\
        done | sed 's/^;//' | sort >.tmp-novell
        (cat in-novell.arpa | sed s/VERSION/`cat install`/;\
         echo '; #### CUT HERE ####' `date` '#### CUT HERE ####',
              cat .tmp-novell) > $(NOVLADDR)
        rm .tmp-novell
```

```
#
# /etc/hosts isn't really used at all, but nice to have
#
$(HOSTS): $(DOMAINS) hosts.private
        @if [ ! -w $(HOSTS) ]; then \
         echo "You need more priviledges\n"; exit 1; \
        fi
        cp hosts.private .tmp-hosts
        echo '#### OFFICAL DATA STARTS HERE ####' >>.tmp-hosts
        echo '# /etc/hosts for uwo.ca sites' >>.tmp-hosts
        echo '#' `date` >>.tmp-hosts
        echo '# Data derived from official DNS data' >>.tmp-hosts
        echo '#' >>.tmp-hosts
        -for i in $(DOMAINS); do\
         grep 100.129.in-addr.arpa $$i;\
        done | sed 's/^;//' |\
        awk -F. '{printf("%s.%s.%s.%s\t%s\n",$$4,$$3,$$2,$$1,$$0)
         awk '{printf("%s\t%s\n",$$1,$$5)}' |\
         sed -e 's/.uwo.ca.$$/.uwo.ca/' | \
         sort >>.tmp-hosts
        install -c -m 644 .tmp-hosts $(HOSTS)
        rm .tmp-hosts
        -if [ -r /etc/mkhosts ]; then\
         /etc/mkhosts $(HOSTS); fi
        -if [ -w $(PUBHOSTS) ]; then\
         sed '1,/#### OFFICAL DATA STARTS HERE ####/d' \
                     $(HOSTS) >$(PUBHOSTS); fi
```



```
#
# The SOA mailing list covers all people who manage domains at We
# those who run their own primary are listed in the file soa.
#
$(SOA):  $(DOMAINS) Makefile soa
        @if [ ! -w $(SOA) ]; then \
         echo "You need more priviledges\n"; exit 1; \
        fi
        grep "@.*SOA" $(DOMAINS) |\
         sed -e 's/.*[      ]\([_a-z0-9]*\).\([._a-z0-9]*\).[
         sort -u > $(SOA)
        cat soa >> $(SOA)
#
# The POSTMAST mailing list covers all people who are postmasters
#
$(POSTMAST):  $(DOMAINS) Makefile postmasters
        @if [ ! -w $(POSTMAST) ]; then \
         echo "You need more priviledges\n"; exit 1; \
        fi
        cp postmasters $(POSTMAST)
        grep -v '^;' *.domain | grep 'MX' |\
         sed 's/.*MX.*[     ]\([._a-z0-9]*\).uwo.ca.*/postmaster@
         sort -u | grep -v void.uwo.ca >> $(POSTMAST)
```

```
#
# After mucking about you need to restart the name daemon and
# update the other name daemons ...
#
install: $(BOOT) $(REVADDR) $(ETHRADDR) $(NOVLADDR) $(HOSTS) $(SC
        @if [ `whoami` != "root" ]; then \
         echo "You need to be root\n"; exit 1; \
        fi
        kill -HUP `cat /etc/named.pid`
        x=`cat install`; x=`expr $$x + 1`; echo $$x >install
        -rm core *~
```