Make and Makefiles[1]

Last Edit
May 29, 1996

Reg Quinton <reggers@julian.uwo.ca>
Information Technology 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 which we use on our Unix systems) 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 (a little bit, not much).

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 licensed product) and RCS (Revision Control System -- this is public domain and the recommended choice of ITS). 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. Read the man page — make(1). Available on **most** Unix systems (or at least those with make).
3. Read more manuals: *Make — A Program for Maintaining Computer Programs*, by S.I. Feldman (in) BSD and System V manuals.
4. Follow the examples set in comp.unix.sources, comp.misc.sources, ftp.uwo.ca:~ftp/pub/unix, 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. This course should be enough to get you started.

Once you have figured out how to use make you might look at imake(1) and xmkmf(1) (which uses imake to build X11 Makefiles). These are tools for making Makefiles! I use these, but I don't know much about them.

1.2. An example

These slides are produced by Make!

```
[1:11pm julian] pwd
/usr/src/usr.local/doc/courses/make
[1:11pm julian] ls
Makefile  RCS       README    sizing    slides
title
[1:11pm julian] page README
These are the slides for my Makefile course. Type
"make" to produce a postscript version suitable
for printing on a laser printer.

        Reg Quinton; 30-Oct-90
[1:11pm julian] make
groff -Tps -me sizing title slides >paper.ps
groff -Tascii -me sizing title slides >paper.txt
```

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

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

and/or

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

1.3. Makefiles

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

```
[11:26am julian] 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] ls
printenv.c
[12:01pm julian] make printenv
        cc  printenv.c  -o printenv
[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: 1992/03/23 19:33:14 $
# $Id: Makefile,v 1.4 1992/03/23 ...
# $Source: /usr/src/usr.local/doc ...
#
# Make everything and install in the right places

SHELL=/bin/sh
SRCS=sizing title slides
DOC=/uwo/ccs/doc/courses/make
MACROS=-me
PSROFF=groff -Tps
NROFF=groff -Tascii
PREVIEW=ghostview

all:	paper.ps paper.txt

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

paper.txt: $(SRCS)
	$(NROFF) $(MACROS) $(SRCS) >paper.txt

preview: paper.ps
	$(PREVIEW) paper.ps
```

```
install: paper.ps paper.txt
	psnup -p2 paper.ps >$(DOC).ps
	install -c -m644 paper.txt $(DOC).txt
	chmod a+r $(DOC).ps $(DOC).txt

clean:; -rm core *~ paper.ps paper.txt
```

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

1.6. Makefile components

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

```
# $Author: reggers $
# $Date: 1992/03/23 19:33:14 $
```

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

```
MACROS=  -me
PSROFF=  groff -Tps
NROFF=   groff -Tascii
```

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

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

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 julian] 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 imported 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 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 — they can be handy. See the target "**make mips**" in my passwd/passwdd suite later in this paper; it calls make with appropriate flags.

A good example of importing a macro from the command line:

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

This would make a version of my slides using ptroff(1) and nroff(1) (for those who don't have groff — the GNU version of troff/nroff).

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
```

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:

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

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

Alternatively, if the install doesn't set the owner, try changing the INSTALL macro:

```
Root: make -n "INSTALL=install -o reggers" install
```

Good Makefiles should be flexible enough that you can do what you need without editting the file.

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

SHELL=/bin/sh
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.

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 special macros used in *implicit rules* (which we will discuss in a moment):

(1) **$<** the files that caused the action (those newer than target).

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

And one final macro you'll need:

(1) **$$** expands to just **$**

For example:

```
users: /etc/passwd
        awk -F: '{print $$1}' /etc/passwd >users
```

We want make to run this command when the password file changes:

```
awk -F: '{print $1}' /etc/passwd >users
```

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!).

If you have no dependents you can use:

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

That's a useful abbreviation — there's nothing fancy going on. But note you'll need the tab!

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) using your SHELL[2]. 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 — who cares if there is no core file?

---

[2] **BEWARE:** Makefiles should always set `SHELL=/bin/sh`. On BSD that was assumed, on System V it's not and will cause problems for /bin/csh users.

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
```

A better example for those who remember teco:

```
love:;      @echo not war
```

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
RANLIB=ranlib

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). On some Unix systems there’s no **ranlib** — use "`make RANLIB=true`".

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:  $(EXES)
        for i in $(EXES); do \
                $(INSTALL) -c -m 755 $$i $(BIN); \
        done
```

Note the use of "**$$i**" and "**$(BIN)**" in:

```
install:  $(EXES)
        for i in $(EXES); do \
                $(INSTALL) -c -m 755 $$i $(BIN); \
        done
```

"**$(BIN)**" we've seen before, that expands to the value of the macro BIN. "**$$**" expands to just "**$**" so, when executed by make, we get

```
for i in files...; do \
       install -c -m 755 $i /usr/local/bin; \
done
```

Ie. we want "**$i**" expanded by the shell, not make. It's a common mistake, be careful.

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. Makefile for printenv(1)

```
# make the printenv command, Reg Quinton.
#
# $Author: reggers $
# $Date: 1994/10/11 14:54:23 $
# $Id: Makefile,v 1.2 1994/10/11 14:54:23 reggers Exp $
# $Source: /usr/internal/nst/reggers/printenv.full/RCS/Makefile,v

SHELL=/bin/sh
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

.c.o:; $(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 $(BINDIR)
	$(INSTALL) -c -o $(OWNER) -g $(GROUP) -m 644 \
		printenv.man $(MANDIR)

# 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. Makefile for Passwd(1)/Passwdd(8)

```
# $Author: reggers $
# $Date: 1994/09/22 13:48:31 $
# $Id: Makefile,v 1.28 1994/09/22 13:48:31 reggers Exp $
# $Source: /usr/src/usr.local/security/passwd-suite/RCS/Makefile,
#
# Make things and install them as well. Original work by Reg Quir
# 24-Dec-91. Much work since then around 13-Sep-94.
#
# If you’re on one of the machines I’ve tested this on -- mips, s
# solaris, aix, or irix you should start with the make targets do
# the tail of this. If you add targets, let me know.

SHELL=/bin/sh

# important directories. I install into /usr/local, you might put
# other places.

TOPDIR=/usr/local
ETC=$(TOPDIR)/etc
BIN=$(TOPDIR)/bin
OLD=$(TOPDIR)/old

# I want the configuration file owned by root and readable by pas
# I use a special group and setgid the passwd image. You might wa
# this to be GROUP=wheel on some systems.

PASSWDRC=/etc/passwdd.rc
```

```
OWNER=root
GROUP=root

# Manual pages, where  are  they

MANDIR=$(TOPDIR)/man
MAN1DIR=$(MANDIR)/man1
MAN1EXT=1
MAN8DIR=$(MANDIR)/man8
MAN8EXT=8

# compilation flags, etc.

CC=cc
CFLAGS=-O
LDFLAGS=-s
LIBS=
INSTALL=install
GET=co

MANS=passwdd.man passwd.man
SRCS=passwdd.c setpw.c crack.c config.c chsh.c chfn.c \
     passwd.c tcpopen.c match.c $(MANS)

DAEMONOBJS=passwdd.o setpw.o crack.o config.o chsh.o chfn.o \
           match.o

CLIENTOBJS=passwd.o tcpopen.o config.o
```

```
EXES=passwdd passwd

all:    $(EXES)

$(SRCS):;       $(GET) $@

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

passwdd:        $(DAEMONOBJS)
       $(CC) $(LDFLAGS) -o $@ $(DAEMONOBJS) $(LIBS)

passwd:         $(CLIENTOBJS)
       $(CC) $(LDFLAGS) -o $@ $(CLIENTOBJS) $(LIBS)

install:        $(EXES) $(MANS)
       -mv $(PASSWDRC) $(ETC)/passwdd \
              $(BIN)/passwd  $(BIN)/chfn $(BIN)/chsh $(OLD)
       $(INSTALL) -c -m  640 -g $(GROUP) -o $(OWNER) \
              passwdd.rc $(PASSWDRC)
       $(INSTALL) -c -m 2755 -g $(GROUP) -o $(OWNER) \
              passwd  $(BIN)
       $(INSTALL) -c -m 755 passwdd $(ETC)
       ln $(BIN)/passwd $(BIN)/chfn
       ln $(BIN)/passwd $(BIN)/chsh
       $(INSTALL) -c -m 644 passwdd.man \
              $(MAN8DIR)/passwdd.$(MAN8EXT)
       $(INSTALL) -c -m 644 passwd.man  \
              $(MAN1DIR)/passwd.$(MAN1EXT)
       ln $(MAN1DIR)/passwd.$(MAN1EXT)  \
```

```
                $(MAN1DIR)/chsh.$(MAN1EXT)
        ln $(MAN1DIR)/passwd.$(MAN1EXT)  \
                $(MAN1DIR)/chfn.$(MAN1EXT)

clean:; -rm core *~ *.o $(EXES)

shar:;  shar -pX `ls | grep -v RCS`

############## makes for various OS flavors ##############
#
# Sunos seems to work for:
# [2:20pm suncon] uname -a
# SunOS suncon.cc 4.1.3_U1 1 sun4c
# [2:20pm suncon] date
# Wed Sep 21 14:20:13 EDT 1994

sunos:
        make CFLAGS='${CFLAGS} -DSETPWFILE'

# Solaris seems to work for:
# [3:12pm falcon] uname -a
# SunOS falcon.ccs.uwo.ca 5.3 Generic_Patch sun4m sparc
# [3:12pm falcon] date
# Wed Sep 21 15:12:16 EDT 1994

solaris:
        make CC='gcc' \
                CFLAGS='${CFLAGS} -I. -Drandom=rand -Dsrandom=sran
                DAEMONOBJS='${DAEMONOBJS} setpwfile.o'\
```

```
                SRCS='${SRCS} setpwfile.c sysexits.h' \
                LIBS='-lsocket -lnsl -lelf'

# Mips seems to work for:
# [2:17pm julian] uname -a
# julian.uwo.ca julian.uwo.ca 1.4.3 UMIPS mips
# [2:17pm julian] date
# Wed Sep 21 14:17:41 EDT 1994

mips:
        make CC='cc -systype bsd43' \
                CFLAGS='${CFLAGS} -DCRYPTOR -DSETPWFILE \
                        -DSIGRET=int -Duid_t=int'\
                DAEMONOBJS='${DAEMONOBJS} cryptor.o'\
                CLIENTOBJS='${CLIENTOBJS} cryptor.o'\
                SRCS='${SRCS} cryptor.c'

# Irix seems to work for:
# [3:05pm mango] uname -a
# IRIX mango 5.2 02282013 IP12 mips
# [3:05pm mango] date
# Wed Sep 21 15:05:12 EDT 1994

irix:
        make CFLAGS='${CFLAGS} -DDICT=
                DAEMONOBJS='${DAEMONOBJS} setpwfile.o getusershell.
                SRCS='${SRCS} setpwfile.c getusershell.c'

# AIX seems to work for:
```

```
# [3:07pm xia] uname -a
# AIX xia 2 3 000065074100
# [3:07pm xia] date
# Wed Sep 21 15:07:56 EDT 1994

aix:
        make CFLAGS=’${CFLAGS} -DSETPWFILE ’ \
                DAEMONOBJS=’${DAEMONOBJS} getusershell.o’\
                SRCS=’${SRCS} getusershell.c’
```

3.2. Makefile Implicit Rules

Consider the rule we used for printenv

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

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 from object files, 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).

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

alternatively

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

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

```
OBJS=x.o y.o z.o

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

program: $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS)
```

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"

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;

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

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), from my printenv command:

```
DEPEND= makedepend $(CFLAGS)

# what are the source dependencies

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

# 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
0       0       * * * \
        cd /usr/local/lib; make aliases
0,15,30,45    * * * * \
        cd /usr/local/lib/named/data; make install
```

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 used to capture these maps when they arrived. This is a pretty classic instance of a file system dependency — data base depends on map files.

```
# 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
SHELL=/bin/sh
# 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 | /bin/sh; \
        done | mail -s "UUCP maps extracted" ${NOTIFY}
        touch 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: 1994/02/21 19:31:55 $
# $Id: Makefile.lib,v 1.23 1994/02/21 19:31:55 reggers Exp $
# $Source: /usr/src/usr.local/lib/RCS/Makefile.lib,v $
#
# Install as /usr/local/lib/Makefile and run periodically for va
# system administration stuff. Original work by Reg Quinton start
# back in 1988.

SHELL=/bin/sh
PATH=/usr/local/bin:/uwo/ccs/bin:/uwo/ccs/share/bin:/usr/ucb:/us

# The sendmail alias file data base. No need to actually do the
# rebuild as sendmail will detect and do it automatically.
# From crontab periodically do:
#      (cd /usr/local/lib; make aliases)

aliases: aliases.usenet aliases.private aliases.users
       cat aliases.usenet \
              aliases.users \
                     aliases.private >aliases
       chmod 644 aliases

# The awk rule is for taking a sequence of records and pulling o
```

```
# those that have a unique name. I don't want common names that
# to different users.

AWKRULE=BEGIN { NAME = ""; ADDR= ""; FS=":";  } \
        { \
                if ( $$1 != NAME) \
                {       if ( ADDR != "" ) \
                                printf("%s:%s\n",NAME,ADDR); \
                        NAME=$$1;       ADDR=$$2; \
                } \
                else \
                if ( $$2 != ADDR) ADDR=""; \
        } \
        END \
        {       if ( ADDR != "" ) printf("%s:%s\n",NAME,ADDR);

# User aliases are from login name and from common name variatio
# This is pretty much the same procedure we use to construct whoi
# names. /etc/passwd entries tacked on the end to assure that lo
# names take precedence.

aliases.users: /etc/passwd
       tr \[A-Z\] \[a-z\] </etc/passwd |\
       sed -n 's/^\([^:]*\):[^:]*:[^:]*:[^:]*: *\([^:,(]*\).*/\2
       sed -n  -e 's/^\([-A-Za-z]\{2,\}\).* \([-A-Za-z]\{2,\}\):
        -e 's/^.* \([-A-Za-z]\{2,\}\).* \([-A-Za-z]\{2,\}\):/\1
        -e 's/^\([-A-Za-z]\).* \([-A-Za-z]\{2,\}\):/\1 \2:/p' \
        -e 's/^.* \([-A-Za-z]\{2,\}\):/\1:/p' |\
       sed -n -e 's/ /./g' -e 's/\.\.*/./g' \
```

```
        -e p -e 's/-[A-Za-z]*:/:/p' -e 's/\([-A-Za-z]\{2,\}\)\..*:
        sort -u  | awk '$(AWKRULE)' >aliases.users
        awk -F: '{printf "%s:%s\n", $$1, $$1}' /etc/passwd | \
                sort >>aliases.users
        chmod 644 aliases.users

# Once a day I go and grab a new copy of the Usenet list of group
# From crontab:
#       (cd /usr/local/lib; make daily)

daily:
        getactive active active
        if [ -s active ]; then \
        awk '{printf "%s: \"|/usr/lib/newsbin/MailNews -n %s\"\n",
                active | sort >aliases.usenet; \
                chmod 644 aliases.usenet; \
        fi
        rm active
```

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

```
# Makefile -- build and install things in the right places
#
# $Author: reggers $
# $Date: 1996/05/29 12:14:43 $
# $Locker:  $
# $Id: slides,v 1.8 1996/05/29 12:14:43 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:/bin
PUBHOSTS=/uwo/ccs/ftp/nic/hosts
HOSTS=/etc/hosts
NETS=/etc/networks
LPDS=/etc/hosts.lpd
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
```

```
# sed format spec for an IP, Novell, and Ethernet numbers

IP=[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}
NOV=8164[0-9A-Fa-f]\{4\}
ETH=[0-9A-Fa-f]\{12\}

# These are sources files managed by RCS. Don’t restart name serv
# any of these are busy!

SRCS=Makefile bitnet.domain bottom.in-addr.arpa ccs.domain cdn.d
       in-addr.arpa in-novell.arpa net.domain uucp.domain uwo.dom
       watcher.ccs.domain slip.domain

DOMAINS= ahs.domain \
       anatomy.domain \
       apmaths.domain \
       arf.domain etc... \
       zoo.domain

all: install

$(DOMAINS):
       co $@

#
# verify that all data files have reasonably accurate data. This
# catches a few of the more common errors. Note that I only check
# DNS data files that have changed since my last check.
#
```

```
check: ${DOMAINS}
       @for i in $?; do\
              if sed -e h \
                     -e ’s/"[^"][^"]*"/word/g’ \
                     -e ’s/^;\([0-9A-Za-z]\)\(.*\)PTR/\1\2PTR/’ \
                     -e ’/(\([^)]*\)$$/N’ \
                     -e ’/(\([^)]*\)$$/N’ \
                     -e ’/(\([^)]*\)$$/N’ \
                     -e ’/(\([^)]*\)$$/N’ \
                     -e ’/(\([^)]*\)$$/N’ \
                     -e ’s/;[^\n]*\n//g’ \
                     -e ’s/;.*$$//’ \
                     -e ’s/\n//g’ \
                     -e ’s/[    ][   ]*/ /g’ \
                     -e ’s/ $$//’ \
                     -e ’/^$$/d’ \
                     -e ’/^$$ORIGIN [A-Za-z.]*\.$$/d’ \
                     -e ’/^$$INCLUDE [^ ]*$$/d’ \
                     -e ’/${IP}\.in-addr\.arpa\. IN PTR [^ ]*\.$$
                     -e ’/${NOV}\.in-novell\.arpa\. IN PTR [^ ]*\
                     -e ’/${ETH}\.in-ether\.arpa\. IN PTR [^ ]*\.
                     -e ’/un.*\.in-ether\.arpa\. IN PTR [^ ]*\.$$
                     -e ’s/^[^ ]*\. IN/ IN/’ \
                     -e ’s/^@ IN/ IN/’ \
                     -e ’/^ IN SOA [^ ]*\. [^ ]*\. /d’  \
                     -e ’/^ IN NS [^ ]*\.$$/d’  \
                     -e ’/^ IN A ${IP}$$/d’  \
                     -e ’/^ IN WKS ${IP} TCP/d’  \
                     -e ’/^ IN WKS ${IP} UDP/d’  \
```

```
                        -e ’/^ IN HINFO [^ "]* [^ "]*$$/d’  \
                        -e ’/^ IN UINFO [^ "]*$$/d’  \
                        -e ’/^ IN CNAME [^ "]*\.$$/d’  \
                        -e ’/^ IN MX [0-9]* [^ ]*\.$$/d’  \
                        -e = -e g $$i|  grep ’^’ ; \
                then\
                        echo 1>&2 Bad/malformed data in $$i; \
                        exit 1; \
                fi; \
                echo DNS data $$i looks clean; \
        done
        date >$@

#
# 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"; exit 1; \
        fi
        @if [ -r .tmp-rev ]; then \
                echo "Oops.. busy, a make must be in progress"; exi
        fi
        -sed -n ’/^;[.0-9]*.in-addr.arpa/s/^;//p’ $(DOMAINS) |\
                        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) > $(REVADD
```

```
        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"; exit 1; \
        fi
        @if [ -r .tmp-ether ]; then \
                echo "Oops.. busy, a make must be in progress"; exi
        fi
        -sed -n ’/^;[.:a-fA-f0-9]*.in-ether.arpa/s/^;//p’ $(DOMAIN
                        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

#
# This one is local, map Novell Netware numbers back to names as
#

$(NOVLADDR): $(DOMAINS) in-novell.arpa
        @if [ ! -w $(NOVLADDR) ]; then \
                echo "You need more privileges"; exit 1; \
        fi
        @if [ -r .tmp-novell ]; then \
                echo "Oops.. busy, a make must be in progress"; exi
```

```
        fi
        -sed -n ’/^;[a-fA-F0-9]*.in-novell.arpa/s/^;//p’ $(DOMAINS
                        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"; exit 1; \
        fi
        @if [ -r .tmp-hosts ]; then \
                echo "Oops.. busy, a make must be in progress"; exi
        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 ’^;[0-9][0-9]*\.[0-9]*\.100\.129\.in-addr.arpa
        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’ $(HOS

#
# /etc/networks, nntp needs this but not much else
#

$(NETS): $(HOSTS) networks.private
        @if [ ! -w $(NETS) ]; then \
                echo "You need more priviledges"; exit 1; \
        fi
        @if [ -r .tmp-networks ]; then \
                echo "Oops.. busy, a make must be in progress"; exi
        fi
        cp networks.private .tmp-networks
        echo ’#### OFFICAL DATA STARTS HERE ####’ >>.tmp-networks
        echo ’# /etc/networks for net.uwo.ca sites’ >>.tmp-network
        echo ’#’ ‘date‘ >>.tmp-networks
        echo ’# Data derived from official DNS data’ >>.tmp-networ
        echo ’#’ >>.tmp-networks
        grep net.uwo.ca $(HOSTS) | sed ’s:\.0::g’ | \
                awk ’{printf("%-25.25s%s\n",$$2,$$1)}’ |\
```

```
                        sort >>.tmp-networks
        install -c -m 644 .tmp-networks $(NETS)
        rm .tmp-networks

#
# /etc/hosts.lpd, we allow anyone to print to us
#

$(LPDS): $(HOSTS)
        @if [ ! -w $(LPDS) ]; then \
                echo "You need more priviledges"; exit 1; \
        fi
        @if [ -r .tmp-lpds ]; then \
                echo "Oops.. busy, a make must be in progress"; exi
        fi
        grep 129.100 $(HOSTS) | awk '{print $$2}' | sort -fu >.tmp
        install -c -m 644 .tmp-lpds $@
        rm .tmp-lpds
#
# 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"; exit 1; \
        fi
        @if [ -r .tmp-soa ]; then \
                echo "Oops.. busy, a make must be in progress"; exi
```

```
        fi
        grep '^@.*SOA' *.domain | awk '{print $$5}' | \
                sed -e 's/\./@/' -e 's/\.$$//' |\
                sort -u > .tmp-soa
        cat soa >> .tmp-soa
        install -c -m 644 .tmp-soa $(SOA)
        rm .tmp-soa
#
# The POSTMAST mailing list covers all people who are postmasters
#

$(POSTMAST): $(DOMAINS) Makefile postmasters
        @if [ ! -w $(POSTMAST) ]; then \
                echo "You need more priviledges"; exit 1; \
        fi
        @if [ -r .tmp-posts ]; then \
                echo "Oops.. busy, a make must be in progress"; exi
        fi
        cp postmasters .tmp-posts
        sed -n 's/^[^;].*MX[     ]*[0-9]*[     ]*\(.*uwo.ca\).*/po
                sort -u | grep -v void.uwo.ca >> .tmp-posts
        install -c -m 644 .tmp-posts $(POSTMAST)
        rm .tmp-posts

# This is a failsafe target to keep one from building things and
# restarting the name server when one of the files managed by RCS
# being worked on. [ -w $i ] doesn't work as root can write any f

busy: $(SRCS)
```

```
        for i in $?; do \
                if /bin/ls -l $$i | awk ’{print $$1}’ | grep w >/de
                        echo $$i is busy, cannot procede; \
                        exit 1 ; \
                fi; \
        done
        touch busy

#
# After mucking about you need to restart the name daemon and
# update the other name daemons ... Note this will bail out if ar
# of the source files are busy or if any of the data files aren’t
# syntactically correct.
#

install: busy check $(BOOT) $(REVADDR) $(ETHRADDR) $(NOVLADDR) \
                $(HOSTS) $(NETS) $(SOA) $(POSTMAST) $(LPDS)
        @if [ ‘whoami‘ != "root" ]; then \
                echo "You need to be root"; exit 1; \
        fi
        kill -HUP ‘cat /etc/named.pid‘
        x=‘cat install‘; x=‘expr $$x + 1‘; echo $$x >install
        -rm core *~
        chmod a+r *
        if tty -s ; then sleep 5; tail /usr/spool/syslog/console;
```