Methods for Maintaining One Source Tree in a Heterogeneous Environment

Bjorn Satdeva
/sys/admin, inc.

Methods for Maintaining One Source Tree in a Heterogeneous Environment

Bjorn Satdeva – /sys/admin, inc.

ABSTRACT

A common problem when maintains software in a heterogeneous environment are the need to maintain executable for several platforms and/or operating system versions. Ideally this should be done from a single source tree. This papers gives a overview of The *Depot, Xheir* and the *make* (1) as it appears in BSD Net2.

Further, the BSD Net2 make is compared to The Depot and Xheir, and some practical experiences with the BSD Net2 make are be presented, and its advantages and shortcomings are outlined.

Introduction

A common problem when maintaining software for a heterogeneous environment are the need to maintain executable for several platforms and/or operating system versions. Ideally this should be done from a single source tree.

Two solutions which partially addresses this problem, *Xheir* and *The Depot* have been presented at previous LISA conferences. However, a third solution, the new BSD *make* (1) first distributed in the BSD 4.3 Tahoe, and now part of the BSD Net 2 release, has received little attention in the LISA community. The BSD Net2 make (hereafter referred to as the Net2 make), has, in my opinion, several very significant advantages over both Xheir and The Depot. The Net2 make do not require use of the automounter, and the symbolic links required (one per directory) are all build and maintained by Net2 make itself. Further, Net2 make knows which platform it is building binaries for, and provides include files and an extended command set, allowing most make files to be only a few lines long.

Although Xheir and The Depot have as their main focus the distribution of compiled software, and they place less emphasis on the development and maintenance issues, it is the only previous work I have found addressing these issues (the *imake* from the X11 distribution addresses maintenance of multiple platforms, but cannot do this simultaneously). This paper gives first a brief overview of source maintenance aspects of Xheir and The Depot, followed by a more complete description of the Net2 make. Finally, some practical examples will be presented of the work which where required for a few software packages in order to use Net2 make to build them, and the tools which can be an aid when maintain the ported software.

Overview of The Depot

The Depot is primary a methodology for distributing software. However, as it provides some for compiling sources for multiple platforms, it has been included here. It uses an elaborate scheme of mount-points and symbolic links to appear as only a single directory tree for a given platform is mounted. It has a limited support for single set of platform independent sources, but there is no evidence that compiles on multiple platforms can occur simultaneously.

The Deport release very heavily on NFS mounts, loopback mounts, and symbolic links, as well as NIS and the SunOS automounter. It appears as a very complex strategy, which attempts to solve simultaneously the handling sources, distributing of executables, and handling of mounting of the binaries for the correct platform. In spite of this, it appears to leave many aspects of a single source tree unresolved.

Overview of Xheir

Xheir is both a development environment and a distribution tool, but as The th Depot, the main emphasis is on the distribution. Like The Depot, it makes the source tree available on different platforms, however, it mainly does so by distribution, rather than sharing through NFS.

The Xheir package has a vary large number of commands (80) and provides a more automated approach to the distribution issue than The Depot. However, like The Depot, it does not address the issues of maintaining a single source tree very well.

Like The Depot, the Xheir package places a high usage on symbolic links, but it does not rely on neither NIS nor the automounter. Like The Depot, the many symbolic links makes the package complex and more difficult to understand. The authors claim that the use of the extended command set enables the maintainers of the various packages to perform their work without understanding the underlying structure. However, it is still required that *somebody* understands it well enough to fix problems when something is broken.

Overview of BSD make

BSD Net2 make is backwards compatible with the previous versions of make. However, it has been redesigned to provide a new higher usability, not found in the previous versions. It differs from The Depot and Xheir in the respect that it is aimed at maintaining a single source tree for multiple platforms while using a single source tree. All object files are kept outside the source tree, enabling the source tree to be NFS mounted read-only on the build machines, if so desired. It is therefore not necessary to jump through hoops to keep the object files for the various platforms when compiling from one source tree. In addition, the Net2 make knows which platform it is running on. This information is kept in the pre-defined variable *${MACHINE}* which can be used either to specify machine type to the compiler through a *-D${MACHINE}* or by using a conditional statement (on this or other variables) at execution time.

To create a directory to keep the object files separate, one needs to type the command *make obj* which will create a directory for the object files (by default below the directory /usr/obj). A symbolic link from the source directory (named *obj*) to the object directory, is built by the Net2 make by typing the command *make obj*. Before starting a compile, make ensures that all files created during the make will therefore be placed in the obj directory. If it is necessary to make explicit references to files in the source directory, it can be done by using the variable *${.CURDIR}*

In a traditional makefile, dependencies must be listed explicit, although some versions of UNIX can create the list of dependencies by way of *makedepend*(1) and insert this information at the end of the makefile. For Net2 make this is no longer necessary, as the command *''make depend''* will call *makedepend*(1) and placed the result in the file *.depend* in the object directory. This not only removes the need for the bad hack of rewriting the makefile, it also allows for each platform keeping its own dependencies, independent of any other platform.

In addition, much of the content we expect in a traditional makefile has been moved into generalized make libraries or include files which are made available by the *.include* command.

However, all this nice functionality comes at a price. In order to to take full advantage of the functionality of BSD Net2 make each program must be located in its own private directory. It is therefore necessary not only to rewrite all makefiles but also to move some, if not all, source files into a new directory structure. However, for most programs, this is only a relative minor task, which can often be done in less than one hour. Overall, the advantages therefore, in my opinion, far outweighs the disadvantages.

An simple example of a Net2 makefile for a single program, is the makefile for the *ps*(1) program. As *ps* consist of only one executable and one man page, it do not require the sub–directories mentioned above. The Net2 makefile for ps is shown in Figure 1.

```
PROG=   ps
SRCS=   devname.c fmt.c keyword.c
SRCS+=  nlist.c print.c ps.c
CFLAGS+=-I/sys
DPADD=  ${LIBMATH} ${LIBKVM} ${LIBUTIL}
LDADD=  -lm -lkvm -lutil
BINGRP= kmem
BINMODE=2555
BINDIR= /bin

.include <bsd.prog.mk>
```

Figure 1: Complete BSD Net2 *Makefile* for ps(1)

In spite of the absence of any visible targets in this Net2 make, there is nevertheless everything necessary to compile all the object modules, link the executable, format the man-page, and install the executable and man page in the correct directories.

Comparisons of The Three Methods

Whether to use The Depot, Xheir, or Net2 make will naturally depend on local practice and preferences. However, I personal much prefer Net2 make over any of the other solutions. The primary reason is that both The Depot and Xheir intermix software development and support with software distribution; two very different problem sets, which best are handled as separate issues. In addition, both appear to be complex to setup and operate. By using Net2 make for the support and development, it is possible to maintain a single source tree for a large number of platforms, with only two requirements: support of symbolic links and a working port of Net2 make No other programs are necessary, no support for the automounter is required, and there is no large system to support.

Because Net2 make does not have the support for distribution, another means must be provided. At all sites where this has been an issues, *Fdist,* has already been in use. This package handles the distribution aspect of software maintenance very well. The use of Fdist for distribution purposed, also provides a solution for packages approach provided in Xheir. However, this has required writing both a include file for Net2 make, and make some accommodations in Fdist previously missing. The solutions provided for this is still work-in-progress, and the entries supporting this have been removed from the Net2 makefiles in the appendices.

One item which curiously appears to be absent in both Xheir and The Depot are the ability to install the software on the build machine, and test it before distributing to wider use. Not all software packages lend themselves well to testing directly in the development location (it might relay on parts of the executables and/or configuration files be found in specific locations). By actually installing the software package and running it on the build machine, better regression testing can be provided, reducing the risk of distributing a faulty software packages. The combination of Net2 make and Fdist allows the person in charge of a given software packages, to build and test the package, and then explicit releasing it to the user community, whenever, the testing has been completed satisfactory.

Xheir provides a facility for adding necessary information to files such as */etc/services* when a new application is added. This is possible a necessary feature at a very large site. However, having system files altered, and possible damaged by automatic installation programs are in my opinion not a desirable solution. With Fdist in place, it is a relatively minor job to modify files such as /etc/services on a site wide basis. Therefore, in the environments described in this paper, no substitute for automatic altering of system files, as provided by Xheir are provided.

The attempt of comparing The Deport and Xheir with Net2 make is a little bit like comparing apples and oranges. However, there is a sufficiently amount of overlap of the problem set that the three tools attempt to solve. In the realm of maintaining software for different platforms, the Net2 make outperforms both Xheir and The Depot by far. With the support of additional tools, such as Fdist and Modules, I find that the sum of maintaining the sources, and make the resulting programs available to the users are less than for either it would be with either The Depot or Xheir.

Porting Sources to BSD Make

As explained above, it is, in addition to provide new make files, necessary to create a new directory structure, and move the various source files into the correct sub-directories. The first few times I did this, the process where somewhat time consuming. However, as the strategy becomes clear, this can be done for many programs in less than hour. Large software packages, such as *c-news* or *smail*3.2 will naturally take a good deal longer.

It has already been shown what the make file for a single executable will look like. However, most packages consist of at least a few different executables. Below is two examples of such conversions. In addition, I will discuss the conversion of *c-news* as an example of a large package.

As a general strategy, I identify each executable, and create a directory for each. I then move all ‘‘dot-c’’ files which is only part of that program into the newly created sub-directory. All include files is moved into a single sub-directory, names ‘‘1include’’, and any leftover ‘‘dot-c’’ files are moved into a directory named ‘‘lib’’. I then build first the library, and the each of the individual executables. When all parts can be build without any errors, they can be installed by the command *make install* and the newly build programs can then be tested on the build machine, before distributed to other users.

Net2 Makefiles for Less

Less where the first package I ever converted. The less package comprises two programs *less* and *lesskey* each with their own man page. Less comes with a configuration script, *linstall*, which builds two platform dependent files, *defines.h* and a *makefile*. A complete and correct port should, in my opinion, include support for this script under Net2 make but this being the first port I did, this was never done. In this case, the *linstall* command was instead moved into a separate directory, *etc*, and the *defines.h* file built as necessary. This file is then moved to another directory, named *include.${MACHINE},* and included in the c-compilers include search path. This is a hack, but it works sufficiently well.

There is a couple of interesting issues illustrated by this makefile. More than one line is necessary to define all the source files for *less.* The += operator concatenates the definitions, so there are no need to escape the newlines. Because this is a local program, it is necessary to override of the default for where the man page shall be installed. This is done by explicitly setting the *MANDIR* variable. Because *less* needs an additional text file with all the help information, an *afterinstall:* target has been added. This will ensure the help file is installed after the executable. The complete makefiles are shown in Appendix A.

Net Makefiles for nntp

The port of nntp (which was done after c-news) took about twenty minutes to complete. The approach where the same as outlined above, but was supported by nntp already having some support for the ‘‘one program, one directory’’ strategy. The support for ‘‘make client’’ and ‘‘make server’’ was maintained.

Some issues which help illustrate the use of Net2 make are the globally included makefile *Makefile.inc*. This file imports the value of *${NEWSBIN}* from c0ws. Also, the two common files, *version.c* and *clientlib.c* are not placed in a library, but are instead found by make, using the

.PATH: directive. The makefiles for nntp are shown in Appendix B.

Experience Porting C-news

Porting c-news to the Net2 Make where a much bigger project, and not only because of the size of package. C-news is split into several sub-directories, with a similar, but not identical, structure as used for the binaries. In addition, it used a number of shell scripts which has to be executed in sequence, by a number of different users (''root'', ''bin'', and ''usenet'). These shell scripts are in turn build by yet another shell script, which builds a number of configuration files.

I have always found this process cumbersome and unattractive, both because of its unusual style and because it do not lend itself well to a large source directory tree, which can be build from the top. I therefore decided, when I had to rewrite the makefiles and the build shell script, I would do it in a manner which I found better suited for my use.

As the result, the build script is rewritten to build just a configuration file for subs (a shell script which is part of the c-news distribution, and a very useful tool in it's own right). All other files which must be created to fit the local customization all build from make. This approach gives the entire make process an interesting twist, as the many of the variables set and used by make are set in a included makefile, built by make. To ensure support for multiple platforms, it was further necessary to place one of the included makefile in the obj directory structure,

The two major problems encountered in doing this port were ensuring all command line defines for the c-compiler were included correctly in the new make files and ensuring that all files where installed in the right place. The first issue where all resolved through the extensive regression tests in c-news, and the latter through trial and error.

Effectively, the time to do this port was about one week. However, much time where also spend on porting the c-news system to the BSDI platform, and to deal with problems created by bugs in the early version of *sh*(1)

The update from the c-news Performance release (using Update, discussed below), to the Performance Plus release took less than one day, including the time to inspect all files which differed between the two releases. This indicates that with appropriate tools, continued maintenance of a software package (once ported to the Net2 make) does not include a inappropriate overhead, but rather, that the time and effort saved by truly having one source tree, is well worth the effort using the Net2 make.

Maintaining Ported Sources

From the very start, it was clear that some support tool was needed for applying updates to software packages, which where distributed with traditional makefiles, but which was locally ported to the Net2 make. There are really two different kinds of problems which needed to be addressed. Updates of minor nature can be handled by Larry Wall's *patch* program; however, once in a while, a program has so many changes that it is resubmitted as complete source rather than as a series of patches.

The support of *patch* had already been resolved by Jeff Polk from BSDI, who readily made his script available. However, the *c-news* package is always released in full, even when the changes are relatively minor. There were two possible avenues to take: either the current port could be discarded and the new release be reported (not a very attractive option) or write a utility which could perform the necessary update by comparing the official released package with the locally supported version, and update those files which differed, The latter approach was chosen, resulting in a perl script currently named *Update* in lieu of a better name.

This tool compares the contents of two directory trees, and attempts, based on file names, to decide where files from the first directory should be moved into the second, in order to make update. The problem with this approach, is that some files in a large distribution, such as *c-news* will have identical file names (think just at files such as Net2 make and *README).* In *Update*, this has been resolved by creating a small table to resolve such conflicts. *Update* first finds all files in both source and destination directory and then starts resolve how files in the source directory should be mapped to files in the destination directory:

1. Resolve mapping of files which full source and destination pathnames are specified in the *Update* mapping file.
2. Resolve mapping of all files whose full path is identical both directory tree's.
3. Map all file names (basename) which are unique in both source and destination directory.
4. Resolve mapping of files which have added an extension in the destination directory (as for example a *Makefile* in the original distribution is renamed to *Makefile.org* in the destination directory
5. Resolve mappings which can be done by specifying a source directory and a destination directory.
6. Repeat mapping of all remaining files whose names now are unique in both directory trees (some ambiguity has probably been removed by now).
7. Acknowledge all ignored files.

Update does not copy any files directly. Instead, it generates a combined report and shell script which includes the necessary commands to copy files which content which differ from the source to the destination directory. This makes it possible to run *Update* as often as necessary, inspect the result, and perform the actual copying only when the output has been deemed correct.

Update was written as a fast hack in less than one day. The result is as could be expected. The commands in the *Update* mapping file are obscure at best. The available commands are shown in Figure 2, the complete map file for c-news are shown in Appendix C.

b	map by base name
D	map source dir to destination dir
e	map by basename plus extension
P	map by path name
i	ignore file
w	ignore path,
I	ignore file, but display warning
W	ignore path, but display warning

Figure 2: Codes used by *Update*

However, the principles used in *Update* have proven themselves very useful. The program has proven to be able to resolve a correct mapping of the 522 files in the c-news distribution, with only 46 explicit mapping instruction in the mapping file. This provides with reasonable assurance that a new version of *c-news* (or any other similar large package), can be ported to our platforms in a reasonable time.

Porting BSD Make to Another System

Net2 make is only useful if it can be used on all system, so sources must be supported. It is therefore necessary, as a prerequisite, to port it to all such systems. The sources for Net2 make comes supposedly with a makefile for this. However, that makefile is completely useless, because a key definition is absent. As part of porting Net2 make itself, it is necessary to decide on a string describing the platform (such as sun3-40 for a Sun3 running SunOS 4.0, or sun4-43, for a Sparc running SunOS 4.3). This name should be defined at compile time in the define MACHINE. In addition, one or more of the functions from the Net 2 distribution which are not available on the target machine (such as *findenv.c, setenv.c,* or *strerror.c*) may need to be copied from the Net 2 sources, and included in the source tree for Net2 make. Appendix D shows parts of a traditional make file (all lines showing the object dependent on the source has been removed to preserve space). This makefile assumes that additional functions necessary to compile the programs on a new platform are in the directory *${MACHINE}.*

Distribution

As mentioned earlier, Net2 make has no means to distribute files; that distribution is a task to be performed separately from software development. By using using a dedicated distribution system, such as *fdist*, to provide the necessary distribution, this can be kept separate from the development system. *Fdist,* as a dedicated distribution system, using a domain oriented addressing scheme, gives a much greater freedom in specifying where files shall be distributed to, without complicating the development environment. From the distribution point of view, the package concept, as seen in *Xheir* are implemented through use of *Fdist* domains, and from the user's perspective, through use of *Modules.*

However, at this time, Net2 make and *Fdist* are not very well integrated, making it a bit frustrating to update the distribution copies of a file. Work is currently in progress to integrate *fdist* and Net2 make. The goal is to enable the developer to type *make distribute* in order to update the currently distributed files.

Conclusion

In my opinion, Net2 make in conjunction with other packages, such as *Fdist* and *Modules*, provides much of the same functionality as found in *The*Depot and *Xheir* with less complexity and fewer requirements to the underlying operating system. As Net2 make is a general software building tool, in the same manner as the tradition *make* program, it also has fewer assumptions about the site, and requires less work to port to new platforms. One could hope, that, as BSD Net2 and BSD 4.4 get more widespread usage, more developers on the net would start to provide some support for Net2 make even if initially, only to support the one program, one directory concept. For people who are required to maintain sources for multiple platforms, this would be an easier world to work in.

Availability

The Net2 Make is part of the BSD Net2 lite distribution, and is available from ftp sites carrying the BSD Net2 sources.

Update where posted to alt.sources beginning of this year. It can also be obtained by sending a request to bjorn@sysadmin.com.

Author Information

Bjorn Satdeva is the President of /sys/admin, inc., a consulting firm which specializes in Large Installation System Administration. Bjorn is also President and founder of Bay-LISA, a San Francisco Bay Area user's group for system administrators of large sites, and Columnist for SysAdmin, the UNIX System Administration Magazine. Contact Bjorn via US mail at /sys/admin, inc.; 2787 Moorpark Avenue;

San Jose, CA 95128; by telephone 408 241-3111, or via e-mail at bjorn@sysadmin.com

References

Manheimnen, K., B. A. Warshaw, S. N. Clark, and W. Rowe, "The Depot: A Framework for Sharing Software Installation Across Organizational and UNIX Platform Boundaries", LISA IV Proceedings, October 1990, pp 37-46.

Wallace Colyer &and Walter Wong, "The Depot: A Tool for Managing Software Environments" LISA VI Proceedings, October 1992, pp 153-159.

John Sellens, "Software Maintenance in a Campus Environment: The Xheir Approach" LISA V Proceedings, October 1991, pp 21-28.

Bjorn Satdeva & Paul Moriarty, "Fdist: A Domain Based, File Distribution System for a Heterogeneous Environment" LISA V Proceedings, October 1991, pp 109-128.

John L. Furlani, "Modules: Providing a Flexible User Environment" LISA V Proceedings, October 1991, pp 141-149.

Make(1) manual page for BSD Net2

bsd.README file from /usr/src/share/mk directory

Appendix A: BSD Net2 Makefiles for *less*

Makefile:

```
SUBDIR =         less lesskey .include <bsd.subdir.mk>
```

less/Makefile

```
PROG=   less
SRCS=   brac.c ch.c charset.c cmdbuf.c command.c decode.c edit.c filename.c
SRCS+=  forwback.c help.c ifile.c input.c jump.c line.c linenum.c lsystem.c
SRCS+=  main.c mark.c optfunc.c option.c opttbl.c os.c output.c position.c
SRCS+=  prompt.c screen.c search.c signal.c tags.c ttyin.c version.c

BINDIR= /usr/local/bin
MANDIR= /usr/local/man/cat
LIBDIR= /usr/local/lib
HELP=   less.hlp

CFLAGS+=-I- -I${.CURDIR}/../include -I${.CURDIR}/../include.${MACHINE}

DPADD+= ${LIBTERM}
LDADD+= -ltermlib

funcs.h:        ${.CURDIR}/../include.${MACHINE}/funcs.h
${.CURDIR}/../include.${MACHINE}/funcs.h:       ${SRCS}
        cd ${.CURDIR}; awk -f mkfuncs.awk ${SRCS} > \
                                        ../include.${MACHINE}/funcs.c

afterinstall:
        install -c -o ${BINOWN} -g ${BINOWN} -m 555 ${HELP} ${LIBDIR}

.include <bsd.prog.mk>
```

lesskey/Makefile

```
PROG=           lesskey
SRCS=           lesskey.c

BINDIR=         /usr/local/bin
MANDIR=         /usr/local/man/cat
CFLAGS+=-I- -I${.CURDIR}/../include -I${.CURDIR}/../include.${MACHINE}

.include <bsd.prog.mk>
```

Appendix B: BSD Net2 Makefiles for *Nntp*

Makefile

```
SUBDIR= doc inews

.ifnmake install
```

```
SUBDIR+= mkgrdates server xmit xfer shlock
.endif

install_server:
        cd server; make install
        cd xmit;   make install
        cd xfer;   make install

.include <bsd.subdir.mk>
```

inews/Makefile

```
.PATH:  ${.CURDIR}/../common ${.CURDIR}/../server
NEWSUSR=usenet
PROG=   inews
SRCS=   inews.c uname.c postauth.c clientlib.c version.c strcasecmp.c
NOMAN=
BINDIR= /usr/local/lib/nntp

install:
        install -c -o ${BINOWN} -g ${BINOWN} -m ${BINMOD} ${PROG} ${BINDIR}

.include <bsd.prog.mk>
```

mkgrdates/Makefile

```
PROG=   mkgrdates
SRCS=   mkgrdates.c
BINDIR= ${NEWSBIN}

.include <bsd.prog.mk>
```

server/Makefile

```
.PROG=   nntpd
SRCS=   main.c serve.c access.c access_inet.c access_dnet.c active.c
SRCS+=  ahbs.c globals.c group.c help.c ihave.c list.c misc.c netaux.c
SRCS+=  newgroups.c newnews.c nextlast.c ngmatch.c post.c parsit.c scandir.c
SRCS+=  slave.c spawn.c strcasecmp.c subnet.c time.c xhdr.c fakesyslog.c
SRCS+=  batch.c auth.c timer.c version.c
BINDIR= /usr/libexec

.include <bsd.prog.mk>
```

xfer/Makefile

```
PROG=    nntpxfer
SRCS=   nntpxfer.c get_tcp_conn.c fakesyslog.c
NOMAN=

.include <bsd.prog.mk>
```

xmit/Makefile

```
PROG=   nntpxmit
SRCS=   nntpxmit.c remote.c llist.c get_tcp_conn.c xmitauth.c
SRCS+=  fakesyslog.c strcasecmp.c

.include <bsd.prog.mk>
```

Appendix C: *Update* mapping file for c-news

```
#       Source                  Destination                     Comment

# Mapping using file name extensions
e       -                       .SH
e       -                       .PROTO

# Files ignored, based on basename
i       -                       Makefile
i       -                       Makefile.inc
```

```
i       -                       README.bsdi
i       -                       CHANGELOG.bsdi

#
g       -                       Makefile
g       -                       Makefile.inc
g       -                       README.bsdi

# Mapping using basename
b       Makefile                Makefile.org
b       makefile                Makefile.org
b       build                   build.org

# The update files
I       inject/rnews            -                       # empty file
I       dbz/dbz.h               -                       # identical to h/dbz.h
I       -                       conf/build              # The new build command

# Directory mapping
D       h                       hfiles/h
D       dbz                     dbz/dbz
D       libbig                  libs/libbig
D       libbsd42                libs/libbsd42
D       libc                    libs/libc
D       libfake                 libs/libfake
D       libcnews                libs/libcnews
D       libsmall                libs/libsmall
D       libstdio                libs/libstdio
D       libusg                  libs/libusg
D       libv7                   libs/libv7
D       libv8                   libs/libv8
D       libusg                  libs/libusg
D       doc                     man/doc
D       man                     man/man

# Duplicate file names which need some guide in resolving
P       explode/morefds.c       explode/explode/morefds.c
P       relay/morefds.c         relay/relaynews/morefds.c

P       explode/trbatch.c       explode/explode/trbatch.c
P       relay/trbatch.c         relay/relaynews/trbatch.c

P       relay/active.c          relay/relaynews/active.c
P       rna/active.c            rna/readnews/active.c

# Special moves,
P       misc/README             maint/README

# Files which are part of makefiles in original distribution
W       -       aux/proto/active.PROTO          Extracted from Makefile.org
W       -       expire/proto/explist.proto      Extracted from expire/Makefile.org
W       -       expire/regress/regress          Extracted from expire/Makefile.org
W       -       dbz/regress/regress             Extracted from dbz/Makefile.org
```

Appendix D – Sample traditional *Makefile* for building Net2 Make

```
MAKE =  /usr/bin/make
OBJS =  arch.o buf.o compat.o dir.o hash.o job.o main.o make.o\
        str.o suff.o targ.o cond.o parse.o var.o

MACHINE=sun4-41
CFLAGS = -O -I. -DMACHINE=\"${MACHINE}\"

all:    machine objs
        cd ${MACHINE}; cc -I. -c *.c
        cd lst.lib; cc -I .. -DMACHINE=\"${MACHINE}\" -c *.c
```

```
        cc *.o lst.lib/*.o ${MACHINE}/*.o -o make

clean:  machine
        cd ${MACHINE}; rm -f *.o
        cd lst.lib; rm -f *.o
        rm -f *.o

objs:   machine ${OBJS}

machine: FRC
        if [ "${MACHINE}" = "" ] ; then  exit 1 ; fi
        if [ ! -d ${MACHINE} ] ; then  exit 1 ; fi

FRC:
```