```
**********************************************************
   10289 Wed Oct 16 17:40:06 2013
new/usr/src/tools/scripts/bldenv.sh
4030 remove useless nightly/bldenv options
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
**********************************************************
_____unchanged_portion_omitted_

 45 function is_source_build
 46 {
 47         "${flags.s.e}" || "${flags.s.d}" || "${flags.s.h}" || "${flags.s.o}"
 48         return $?
 49 }

 51 #
 52 # single function for setting -S flag and doing error checking.
 53 # usage: set_S_flag <type>
 54 # where <type> is the source build type ("E", "D", ...).
 55 #
 56 function set_S_flag
 57 {
 58         if is_source_build; then
 59                 print 'Can only build one source variant at a time.'
 60                 exit 1
 61         fi
 62
 63         case "$1" in
 64                 "E") flags.s.e=true ;;
 65                 "D") flags.s.d=true ;;
 66                 "H") flags.s.h=true ;;
 67                 "O") flags.s.o=true ;;
 68                 *)   usage ;;
 69         esac
 70 }

 45 typeset -r USAGE=$'+
 46 [-?\n@(#)\$Id: bldenv (OS/Net) 2008-04-06 \$\n]
 47 [-author?OS/Net community <tools-discuss@opensolaris.org>]
 48 [+NAME?bldenv - spawn shell for interactive incremental OS-Net
 49     consolidation builds]
 50 [+DESCRIPTION?bldenv is a useful companion to the nightly(1) script for
 51     doing interactive and incremental builds in a workspace
 52     already built with nightly(1). bldenv spawns a shell set up
 53     with the same environment variables taken from an env_file,
 54     as prepared for use with nightly(1).]
 55 [+?In addition to running a shell for interactive use, bldenv
 56     can optionally run a single command in the given environment,
 57     in the vein of sh -c or su -c. This is useful for
 58     scripting, when an interactive shell would not be. If the
 59     command is composed of multiple shell words or contains
 60     other shell metacharacters, it must be quoted appropriately.]
 61 [+?bldenv is particularly useful for testing Makefile targets
 62     like clobber, install and _msg, which otherwise require digging
 63     through large build logs to figure out what is being
 64     done.]
 92 [+?bldenv is also useful if you run into build issues with the
 93     source product or when generating OpenSolaris deliverables.
 94     If a source product build is flagged, the environment is set
 95     up for building the indicated source product tree, which is
 96     assumed to have already been created. If the OpenSolaris
 97     deliverables flag (-O) is set in NIGHTLY_OPTIONS, the
 98     environment is set up for building just the open source.
 99     This includes using an alternate proto area, as well as
100     using the closed binaries in $CODEMGR_WS/closed.skel (which
101     is assumed to already exist).]
 65 [+?By default, bldenv will invoke the shell specified in
```



```
 66     $SHELL. If $SHELL is not set or is invalid, csh will be
 67     used.]
 68 [c?force the use of csh, regardless of the  value  of $SHELL.]
 69 [f?invoke csh with the -f (fast-start) option. This option is valid
 70     only if $SHELL is unset or if it points to csh.]
 71 [d?set up environment for doing DEBUG builds (default is non-DEBUG)]
 72 [t?set up environment to use the tools in usr/src/tools (this is the
 73     default, use +t to use the tools from /opt/onbld)]
111 [S]:[option?Build a variant of the source product.
112 The value of \aoption\a must be one of the following:]{
113         [+E?build the exportable source variant of the source product.]
114         [+D?build the domestic  source  (exportable + crypt) variant of
115             the source product.]
116         [+H?build hybrid source (binaries + deleted source).]
117         [+O?simulate an OpenSolaris (open source only) build.]
118 }

 75 <env_file> [command]

 77 [+EXAMPLES]{
 78     [+?Example 1: Interactive use]{
 79         [+?Use bldenv to spawn a shell to perform  a  DEBUG  build  and
 80             testing of the  Makefile  targets  clobber and install for
 81             usr/src/cmd/true.]
 82         [+\n% rlogin wopr-2 -l gk
 83 {root::wopr-2::49} bldenv -d /export0/jg/on10-se.env
 84 Build type   is  DEBUG
 85 RELEASE      is  5.10
 86 VERSION      is  wopr-2::on10-se::11/01/2001
 87 RELEASE_DATE is  May 2004
 88 The top-level 'setup\' target is available to build headers
 89 and tools.
 90 Using /usr/bin/tcsh as shell.
 91 {root::wopr-2::49}
 92 {root::wopr-2::49} cd $SRC/cmd/true
 93 {root::wopr-2::50} make
 94 {root::wopr-2::51} make clobber
 95 /usr/bin/rm -f true true.po
 96 {root::wopr-2::52} make
 97 /usr/bin/rm -f true
 98 cat true.sh > true
 99 chmod +x true
100 {root::wopr-2::53} make install
101 install -s -m 0555 -u root -g bin -f /export0/jg/on10-se/proto/root_sparc/usr/bi
102 'install\' is up to date.]
103     }
104     [+?Example 2: Non-interactive use]{
105         [+?Invoke bldenv to create SUNWonbld with a single command:]
106         [+\nexample% bldenv onnv_06 \'cd $SRC/tools && make pkg\']
107         }
108 }
109 [+SEE ALSO?\bnightly\b(1)]
110 '

112 # main
113 builtin basename

115 # boolean flags (true/false)
116 typeset flags=(
117         typeset c=false
118         typeset f=false
119         typeset d=false
120         typeset O=false
121         typeset o=false
122         typeset t=true
123         typeset s=(
```

```
124                 typeset e=false
125                 typeset h=false
126                 typeset d=false
127                 typeset o=false
128         )
129 )

131 typeset progname="$(basename -- "${0}")"

133 OPTIND=1
134 SUFFIX="-nd"

136 while getopts -a "${progname}" "${USAGE}" OPT ; do
137     case ${OPT} in
138           c)    flags.c=true  ;;
139           +c)   flags.c=false ;;
140           f)    flags.f=true  ;;
141           +f)   flags.f=false ;;
142           d)    flags.d=true  SUFFIX=""    ;;
143           +d)   flags.d=false SUFFIX="-nd" ;;
144           t)    flags.t=true  ;;
145           +t)   flags.t=false ;;
191           S)    set_S_flag "$OPTARG" ;;
146           \?)   usage ;;
147     esac
148 done
149 shift $((OPTIND-1))

151 # test that the path to the environment-setting file was given
152 if (( $# < 1 )) ; then
153         usage
154 fi

156 # force locale to C
157 export \
158         LC_COLLATE=C \
159         LC_CTYPE=C \
160         LC_MESSAGES=C \
161         LC_MONETARY=C \
162         LC_NUMERIC=C \
163         LC_TIME=C

165 # clear environment variables we know to be bad for the build
166 unset \
167         LD_OPTIONS \
168         LD_LIBRARY_PATH \
169         LD_AUDIT \
170         LD_BIND_NOW \
171         LD_BREADTH \
172         LD_CONFIG \
173         LD_DEBUG \
174         LD_FLAGS \
175         LD_LIBRARY_PATH_64 \
176         LD_NOVERSION \
177         LD_ORIGIN \
178         LD_LOADFLTR \
179         LD_NOAUXFLTR \
180         LD_NOCONFIG \
181         LD_NODIRCONFIG \
182         LD_NOOBJALTER \
183         LD_PRELOAD \
184         LD_PROFILE \
185         CONFIG \
186         GROUP \
187         OWNER \
188         REMOTE \
```



```
189         ENV \
190         ARCH \
191         CLASSPATH

193 #
194 # Setup environment variables
195 #
196 if [[ -f /etc/nightly.conf ]]; then
197         source /etc/nightly.conf
198 fi

200 if [[ -f "$1" ]]; then
201         if [[ "$1" == */* ]]; then
202                 source "$1"
203         else
204                 source "./$1"
205         fi
206 else
207         if [[ -f "/opt/onbld/env/$1" ]]; then
208                 source "/opt/onbld/env/$1"
209         else
210                 printf \
211                     'Cannot find env file as either %s or /opt/onbld/env/%s\n' \
212                     "$1" "$1"
213                 exit 1
214         fi
215 fi
216 shift

218 # contents of stdenv.sh inserted after next line:
219 # STDENV_START
220 # STDENV_END

222 # Check if we have sufficient data to continue...
223 [[ -v CODEMGR_WS ]] || fatal_error "Error: Variable CODEMGR_WS not set."
224 [[ -d "${CODEMGR_WS}" ]] || fatal_error "Error: ${CODEMGR_WS} is not a directory
225 [[ -f "${CODEMGR_WS}/usr/src/Makefile" ]] || fatal_error "Error: ${CODEMGR_WS}/u

227 # must match the getopts in nightly.sh
228 OPTIND=1
229 NIGHTLY_OPTIONS="-${NIGHTLY_OPTIONS#-}"
230 while getopts '+0aBCDdFfGIilMmNnopRrtUuWwXxz' FLAG "$NIGHTLY_OPTIONS"
276 while getopts '+0aBCDdFfGIilMmNnOopRrS:tUuWwXxz' FLAG "$NIGHTLY_OPTIONS"
231 do
232         case "$FLAG" in
279           O)    flags.O=true  ;;
280           +O)   flags.O=false ;;
233           o)    flags.o=true  ;;
234           +o)   flags.o=false ;;
235           t)    flags.t=true  ;;
236           +t)   flags.t=false ;;
285           S)    set_S_flag "$OPTARG" ;;
237           *)    ;;
238         esac
239 done

241 POUND_SIGN="#"
242 # have we set RELEASE_DATE in our env file?
243 if [ -z "$RELEASE_DATE" ]; then
244         RELEASE_DATE=$(LC_ALL=C date +"%B %Y")
245 fi
246 BUILD_DATE=$(LC_ALL=C date +%Y-%b-%d)
247 BASEWSDIR=$(basename -- "${CODEMGR_WS}")
248 DEV_CM="\"@(#)SunOS Internal Development: $LOGNAME $BUILD_DATE [$BASEWSDIR]\""
249 export DEV_CM RELEASE_DATE POUND_SIGN
```

```
export INTERNAL_RELEASE_BUILD=

print 'Build type   is  \c'
if ${flags.d} ; then
        print 'DEBUG'
        unset RELEASE_BUILD
        unset EXTRA_OPTIONS
        unset EXTRA_CFLAGS
else
        # default is a non-DEBUG build
        print 'non-DEBUG'
        export RELEASE_BUILD=
        unset EXTRA_OPTIONS
        unset EXTRA_CFLAGS
fi

[[ "${flags.O}" == "true" ]] && export MULTI_PROTO="yes"

# update build-type variables
PKGARCHIVE="${PKGARCHIVE}${SUFFIX}"

# Append source version
if "${flags.s.e}" ; then
        VERSION+=":EXPORT"
        SRC="${EXPORT_SRC}/usr/src"
fi

if "${flags.s.d}" ; then
        VERSION+=":DOMESTIC"
        SRC="${EXPORT_SRC}/usr/src"
fi

if "${flags.s.h}" ; then
        VERSION+=":HYBRID"
        SRC="${EXPORT_SRC}/usr/src"
fi

if "${flags.s.o}" ; then
        VERSION+=":OPEN_ONLY"
        SRC="${OPEN_SRCDIR}/usr/src"
fi

#       Set PATH for a build
PATH="/opt/onbld/bin:/opt/onbld/bin/${MACH}:/opt/SUNWspro/bin:/usr/ccs/bin:/usr/
if [[ "${SUNWSPRO}" != "" ]]; then
        export PATH="${SUNWSPRO}/bin:$PATH"
fi

TOOLS="${SRC}/tools"
TOOLS_PROTO="${TOOLS}/proto/root_${MACH}-nd" ; export TOOLS_PROTO

if "${flags.t}" ; then
        export ONBLD_TOOLS="${ONBLD_TOOLS:=${TOOLS_PROTO}/opt/onbld}"

        export STABS="${TOOLS_PROTO}/opt/onbld/bin/${MACH}/stabs"
        export CTFSTABS="${TOOLS_PROTO}/opt/onbld/bin/${MACH}/ctfstabs"
        export GENOFFSETS="${TOOLS_PROTO}/opt/onbld/bin/genoffsets"

        export CTFCONVERT="${TOOLS_PROTO}/opt/onbld/bin/${MACH}/ctfconvert"
        export CTFMERGE="${TOOLS_PROTO}/opt/onbld/bin/${MACH}/ctfmerge"

        export CTFCVTPTBL="${TOOLS_PROTO}/opt/onbld/bin/ctfcvtptbl"
        export CTFFINDMOD="${TOOLS_PROTO}/opt/onbld/bin/ctffindmod"

        PATH="${TOOLS_PROTO}/opt/onbld/bin/${MACH}:${PATH}"
        PATH="${TOOLS_PROTO}/opt/onbld/bin:${PATH}"
```



```
        export PATH
fi

export DMAKE_MODE=${DMAKE_MODE:-parallel}

if "${flags.o}" ; then
        export CH=
else
        unset CH
fi
DEF_STRIPFLAG="-s"

TMPDIR="/tmp"

# "o_FLAG" is used by "nightly.sh" (it may be useful to rename this
# variable using a more descriptive name later)
export o_FLAG="$(${flags.o} && print 'y' || print 'n')"

export \
        PATH TMPDIR \
        POUND_SIGN \
        DEF_STRIPFLAG \
        RELEASE_DATE
unset \
        CFLAGS \
        LD_LIBRARY_PATH

# a la ws
ENVLDLIBS1=
ENVLDLIBS2=
ENVLDLIBS3=
ENVCPPFLAGS1=
ENVCPPFLAGS2=
ENVCPPFLAGS3=
ENVCPPFLAGS4=
PARENT_ROOT=
PARENT_TOOLS_ROOT=

if [[ "$MULTI_PROTO" != "yes" && "$MULTI_PROTO" != "no" ]]; then
        printf \
            'WARNING: invalid value for MULTI_PROTO (%s); setting to "no".\n' \
            "$MULTI_PROTO"
        export MULTI_PROTO="no"
fi

[[ "$MULTI_PROTO" == "yes" ]] && export ROOT="${ROOT}${SUFFIX}"

if "${flags.O}" ; then
        print "OpenSolaris closed binary generation requires "
        print "closed tree"
        exit 1
fi

ENVLDLIBS1="-L$ROOT/lib -L$ROOT/usr/lib"
ENVCPPFLAGS1="-I$ROOT/usr/include"
MAKEFLAGS=e

export \
        ENVLDLIBS1 \
        ENVLDLIBS2 \
        ENVLDLIBS3 \
        ENVCPPFLAGS1 \
        ENVCPPFLAGS2 \
        ENVCPPFLAGS3 \
        ENVCPPFLAGS4 \
        MAKEFLAGS \
```

```
354         PARENT_ROOT \
355         PARENT_TOOLS_ROOT

357 printf 'RELEASE      is %s\n'   "$RELEASE"
358 printf 'VERSION      is %s\n'   "$VERSION"
359 printf 'RELEASE_DATE is %s\n\n' "$RELEASE_DATE"

361 if [[ -f "$SRC/Makefile" ]] && egrep -s '^setup:' "$SRC/Makefile" ; then
362         print "The top-level 'setup' target is available \c"
363         print "to build headers and tools."
364         print ""

366 elif "${flags.t}" ; then
367         printf \
368             'The tools can be (re)built with the install target in %s.\n\n' \
369             "${TOOLS}"
370 fi

372 #
373 # place ourselves in a new task, respecting BUILD_PROJECT if set.
374 #
375 /usr/bin/newtask -c $$ ${BUILD_PROJECT:+-p$BUILD_PROJECT}

377 if [[ "${flags.c}" == "false" && -x "$SHELL" && \
378     "$(basename -- "${SHELL}")" != "csh" ]]; then
379         # $SHELL is set, and it's not csh.

381         if "${flags.f}" ; then
382                 print 'WARNING: -f is ignored when $SHELL is not csh'
383         fi

385         printf 'Using %s as shell.\n' "$SHELL"
386         exec "$SHELL" ${@:+-c "$@"}

388 elif "${flags.f}" ; then
389         print 'Using csh -f as shell.'
390         exec csh -f ${@:+-c "$@"}

392 else
393         print 'Using csh as shell.'
394         exec csh ${@:+-c "$@"}
395 fi

397 # not reached
```

```
**********************************************************
   17797 Wed Oct 16 17:40:06 2013
new/usr/src/tools/scripts/nightly.1
4030 remove useless nightly/bldenv options
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
**********************************************************
```

```
.\" "
.\" " The contents of this file are subject to the terms of the
.\" " Common Development and Distribution License (the "License").
.\" " You may not use this file except in compliance with the License.
.\" "
.\" " You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
.\" " or http://www.opensolaris.org/os/licensing.
.\" " See the License for the specific language governing permissions
.\" " and limitations under the License.
.\" "
.\" " When distributing Covered Code, include this CDDL HEADER in each
.\" " file and include the License file at usr/src/OPENSOLARIS.LICENSE.
.\" " If applicable, add the following below this CDDL HEADER, with the
.\" " fields enclosed by brackets "[]" replaced with your own identifying
.\" " information: Portions Copyright [yyyy] [name of copyright owner]
.\" "
.\" " CDDL HEADER END
.\" "
.\" "Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved
.\" "Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
.\" "
.TH nightly 1 "6 July 2010"
.SH NAME
.I nightly
\- build an OS-Net consolidation overnight
.SH SYNOPSIS
\fBnightly [-in] [-V VERS] <env_file>\fP
.LP
.SH DESCRIPTION
.IX "OS-Net build tools" "nightly" "" "\fBnightly\fP"
.LP
.I nightly,
the mother of all build scripts,
can bringover, build, archive, package, error check, and
generally do everything it takes to
turn OS/Net consolidation source code into useful stuff.
It is customizable to permit you to run anything from a
simple build to all of the cross-checking a gatekeeper
needs.  The advantage to using
.I nightly
is that you build things correctly, consistently and
automatically, with the best practices; building with
.I nightly
can mean never having to say you’re sorry to your
gatekeeper.
.LP
More
specifically,
.I nightly
performs the following tasks, in order, if
all these things are desired:
.LP
.RS
.TP
\(bu
perform a "make clobber" to clean up old binaries
.TP
\(bu
bringover from the identified parent gate/clone
.TP
```



```
\(bu
perform non-DEBUG and DEBUG builds
.TP
\(bu
list proto area files and compare with previous list
.TP
\(bu
copy updated proto area to parent
.TP
\(bu
list shared lib interface and compare with previous list
.TP
\(bu
perform a "make lint" of the kernel and report errors
.TP
\(bu
perform a "make check" to report hdrchk/cstyle errors
.TP
\(bu
report the presence of any core files
.TP
\(bu
check the ELF runtime attributes of all dynamic objects
.TP
\(bu
check for unreferenced files
.TP
\(bu
report on which proto area objects have changed (since the last build)
.TP
\(bu
report the total build time
.TP
\(bu
save a detailed log file for reference
.TP
\(bu
mail the user a summary of the completed build
.RE
.LP
The actions of the script are almost completely determined by
the environment variables in the
.I env
file, the only necessary argument.  Ths only thing you really
need to use
.I nightly
is an
.I env
file that does what you want.
.LP
Like most of the other build tools in usr/src/tools, this script tends
to change on a fairly regular basis; do not expect to be able to build
OS/Net with a version of nightly significantly older than your source
tree.  It has what is effectively a Consolidation Private relationship
to other build tools and with many parts of the OS/Net makefiles,
although it may also be used to build other consolidations.
.LP
.SH NIGHTLY_OPTIONS
The environment variable NIGHTLY_OPTIONS controls the actions
.I nightly
will take as it proceeds.
The -i, -n, +t and -V options may also be used from the command
line to control the actions without editing your environment file.
The -i and -n options complete the build more quickly by bypassing
some actions. If NIGHTLY_OPTIONS is not set, then "-Bmt" build
options will be used.
```

```
.B Basic action options
.TP 10
.B \-D
Do a build with DEBUG on (non-DEBUG is built by default)
.TP
.B \-F
Do _not_ do a non-DEBUG build (use with -D to get just a DEBUG build)
.TP
.B \-M
Do not run pmodes (safe file permission checker)
.TP
.B \-i
Do an incremental build, suppressing the "make clobber" that by
default removes all existing binaries and derived files.  From the
command line, -i also suppresses the lint pass and the cstyle/hdrchk
pass
.TP
.B \-n
Suppress the bringover so that the build will start immediately with
current source code
.TP
.B \-o
Do an "old style" (pre-S10) build using root privileges to set OWNER
and GROUP from the Makefiles.
.TP
.B \-p
Create packages for regular install
.TP
.B \-U
Update proto area in the parent workspace
.TP
.B \-u
Update the parent workspace with files generated by the build, as follows.
.RS
.TP
\(bu
Copy proto_list_${MACH} and friends to usr/src in the parent.
.TP
\(bu
When used with -f, build a usr/src/unrefmaster.out in
the parent by merging all the usr/src/unref-${MACH}.out files in the
parent.
.TP
\(bu
When used with -A or -r, copy the contents of the resulting
ELF-data.${MACH} directory to usr/src/ELF-data.${MACH} in the parent
workspace.
.RE
.TP
.B \-m
Send mail to $MAILTO at end of build
.TP
.B \-t
Build and use the tools in $SRC/tools (default setting).
.TP
.B \+t
Use the build tools in "$ONBLD_TOOLS/bin".

.LP
.B Code checking options
.TP 10
.B \-A
Check for ABI discrepancies in .so files.
It is only required for shared object developers when there is an
addition, deletion or change of interface in the .so files.
```



```
.TP
.B \-C
Check for cstyle/hdrchk errors
.TP
.B \-f
Check for unreferenced files.  Since the full workspace must be built
in order to accurately identify unreferenced files, -f is ignored for
incremental (-i) builds, or builds that do not include -l, and -p.
.TP
.B \-r
Check the ELF runtime attributes of all dynamic objects
.TP
.B \-l
Do "make lint" in $LINTDIRS (default: $SRC n)
.TP
.B \-N
Do not run protocmp or checkpaths (note: this option is not
recommended, especially in conjunction with the \-p option)
.TP
.B \-W
Do not report warnings (for freeware gate ONLY)
.TP
.B \-w
Report which proto area objects differ between this and the last build.
See wsdiff(1) for details. Note that the proto areas used for comparison
are the last ones constructed as part of the build. As an example, if both
a non-debug and debug build are performed (in that order), then the debug
proto area will be used for comparison (which might not be what you want).
.LP
.B Groups of options
.TP 10
.B \-G
Gate keeper default group of options (-u)
.TP
.B \-I
Integration engineer default group of options (-mpu)
.TP
.B \-R
Default group of options for building a release (-mp)

.LP
.B Source Build options
.TP 10
.B \-S E | D | H
Build the Export, Domestic, or Hybrid source product. Only Export and
Domestic are truly buildable at this time.
.TP 10
.B \-S O
Simulate an OpenSolaris build on a full tree.  This can be used by
internal developers to ensure that they haven't broken the build for
external developers.
.LP
Source build options only make sense for a full internal tree (open
and closed source).  Only one source build option can be specified at
a time.

.LP
.B Miscellaneous options
.TP 10
.B \-O
generate deliverables for OpenSolaris.  Tarballs containing signed
cryptographic binaries and binaries
of closed-source components are put in $CODEMGR_WS.  (The
cryptographic tarballs are copies of the
ones that are put in the parent directory of
$PKGARCHIVE.)
```

```
.TP 10
.B \-V VERS
set the build version string to VERS, overriding VERSION
.TP
.B \-X
Copies the proto area and packages from the IHV and IHV-bin gates into the
nightly proto and package areas.  This is only available on i386.  See
.B REALMODE ENVIRONMENT VARIABLES
and
.B BUILDING THE IHV WORKSPACE
below.

.LP
.SH ENVIRONMENT VARIABLES
.LP
Here is a list of prominent environment variables that
.I nightly
references and the meaning of each variable.
.LP
.RE
.B CODEMGR_WS
.RS 5
The root of your workspace, including whatever metadata is kept by
the source code management system.  This is the workspace in which the
build will be done.
.RE
.LP
.B PARENT_WS
.RS 5
The root of the workspace that is the parent of the
one being built.  This is particularly relevant for configurations
with a main
workspace and build workspaces underneath it; see the
\-u and \-U
options as well as the PKGARCHIVE environment variable, for more
information.
.RE
.LP
.B BRINGOVER_WS
.RS 5
This is the workspace from which
.I nightly
will fetch sources to either populate or update your workspace;
it defaults to $CLONE_WS.
.RE
.LP
.B CLOSED_BRINGOVER_WS
.RS 5
A full Mercurial workspace has two repositories: one for open source
and one for closed source.  If this variable is non-null,
.I nightly
will pull from the repository that it names to get the closed source.
It defaults to $CLOSED_CLONE_WS.
.LP
If $CODEMGR_WS already exists and contains only the open repository,
.I nightly
will ignore this variable; you'll need to pull the closed repository
by hand if you want it.
.RE
.LP
.B CLONE_WS
.RS 5
This is the workspace from which
.I nightly
will fetch sources by default.  This is
often distinct from the parent, particularly if the parent is a gate.
```



```
.RE
.LP
.B CLOSED_CLONE_WS
.RS 5
This is the default closed-source Mercurial repository that
.I nightly
might pull from (see
.B CLOSED_BRINGOVER_WS
for details).
.RE
.LP
.B SRC
.RS 5
Root of OS-Net source code, referenced by the Makefiles.  It is
the starting point of build activity.  It should be expressed
in terms of $CODEMGR_WS.
.RE
.LP
.B ROOT
.RS 5
Root of the proto area for the build.  The makefiles direct
installation of build products to this area and
direct references to these files by builds of commands and other
targets.  It should be expressed in terms of $CODEMGR_WS.
.LP
If $MULTI_PROTO is "no", $ROOT may contain a DEBUG or non-DEBUG
build.  If $MULTI_PROTO is "yes", $ROOT contains the DEBUG build and
$ROOT-nd contains the non-DEBUG build.
.LP
For OpenSolaris deliveries (\fB\-O\fR), $ROOT-closed contains a parallel
proto area containing the DEBUG build of just usr/closed components, and
$ROOT-nd-closed contains the non-DEBUG equivalent.
.RE
.LP
.B TOOLS_ROOT
.RS 5
Root of the tools proto area for the build.  The makefiles direct
installation of tools build products to this area.  Unless \fB+t\fR
is part of $NIGHTLY_OPTIONS, these tools will be used during the
build.
.LP
As built by nightly, this will always contain non-DEBUG objects.
Therefore, this will always have a -nd suffix, regardless of
$MULTI_PROTO.
.RE
.LP
.B MACH
.RS 5
The instruction set architecture of the build machine as given
by \fIuname -p\fP, e.g. sparc, i386.
.RE
.LP
.B LOCKNAME
.RS 5
The name of the file used to lock out multiple runs of
.IR nightly .
This should generally be left to the default setting.
.RE
.LP
.B ATLOG
.RS 5
The location of the log directory maintained by
.IR nightly .
This should generally be left to the default setting.
.RE
.LP
```

```
.B LOGFILE
.RS 5
The name of the log file in the $ATLOG directory maintained by
.IR nightly .
This should generally be left to the default setting.
.RE
.LP
.B STAFFER
.RS 5
The non-root account to use on the build machine for the
bringover from the clone or parent workspace.
This may not be the same identify used by the SCM.
.RE
.LP
.B MAILTO
.RS 5
The address to be used to send completion e-mail at the end of
the build (for the \-m option).
.RE
.LP
.B MAILFROM
.RS 5
The address to be used for From: in the completion e-mail at the
end of the build (for the \-m option).
.RE
.LP
.B REF_PROTO_LIST
.RS 5
Name of file used with protocmp to compare proto area contents.
.RE
.LP
.B PARENT_ROOT
.RS 5
The parent root, which is the destination for copying the proto
area(s) when using the \-U option.
.RE
.LP
.B PARENT_TOOLS_ROOT
.RS 5
The parent tools root, which is the destination for copying the tools
proto area when using the \-U option.
.RE
.LP
.B RELEASE
.RS 5
The release version number to be used; e.g., 5.10.1 (Note: this is set
in Makefile.master and should not normally be overridden).
.RE
.LP
.B VERSION
.RS 5
The version text string to be used; e.g., "onnv:‘date ’+%Y-%m-%d’’".
.RE
.LP
.B RELEASE_DATE
.RS 5
The release date text to be used; e.g., October 2009. If not set in
your environment file, then this text defaults to the output from
$(LC_ALL=C date +"%B %Y"); e.g., "October 2009".
.RE
.LP
.B INTERNAL_RELEASE_BUILD
.RS 5
See Makefile.master - but it mostly controls id strings. Generally,
let
.I nightly
```



```
set this for you.
.RE
.LP
.B RELEASE_BUILD
.RS 5
Define this to build a release with a non-DEBUG kernel.
Generally, let
.I nightly
set this for you based on its options.
.RE
.LP
.B PKGARCHIVE
.RS 5
The destination for packages.  This may be relative to
$CODEMGR_WS for private packages or relative to $PARENT_WS
if you have different workspaces for different architectures
but want one hierarchy of packages.
.RE
.LP
.B MAKEFLAGS
.RS 5
Set default flags to make; e.g., -k to build all targets regardless of errors.
.RE
.LP
.B UT_NO_USAGE_TRACKING
.RS 5
Disables usage reporting by listed Devpro tools. Otherwise it sends mail
to some Devpro machine every time the tools are used.
.RE
.LP
.B LINTDIRS
.RS 5
Directories to lint with the \-l option.
.RE
.LP
.B BUILD_TOOLS
.RS 5
BUILD_TOOLS is the root of all tools including the compilers; e.g.,
/ws/onnv-tools.  It is used by the makefile system, but not nightly.
.RE
.LP
.B ONBLD_TOOLS
.RS 5
ONBLD_TOOLS is the root of all the tools that are part of SUNWonbld; e.g.,
/ws/onnv-tools/onbld.  By default, it is derived from
.BR BUILD_TOOLS .
It is used by the makefile system, but not nightly.
.RE
.LP
.B SPRO_ROOT
.RS 5
The gate-defined default location for the Sun compilers, e.g.
/ws/onnv-tools/SUNWspro.  By default, it is derived from
.BR BUILD_TOOLS .
It is used by the makefile system, but not nightly.
.RE
.LP
.B JAVA_ROOT
.RS 5
The location for the java compilers for the build, generally /usr/java.
.RE
.LP
.B OPTHOME
.RS 5
The gate-defined default location of things formerly in /opt; e.g.,
/ws/onnv-tools.  This is used by nightly, but not the makefiles.
```

```
.RE
.LP
.B TEAMWARE
.RS 5
The gate-defined default location for the Teamware tools; e.g.,
/ws/onnv-tools/SUNWspro.  By default, it is derived from
.BR OPTHOME .
This is used by nightly, but not the makefiles.  There is no
corresponding variable for Mercurial or Subversion, which are assumed
to be installed in the default path.
.RE
.LP
.B OPEN_SRCDIR
.RS 5
The open source tree is copied to this directory when simulating an
OpenSolaris build (\fB\-S O\fR).  It defaults to $CODEMGR_WS/open_src.
.LP
.RE
.B ON_CLOSED_BINS
.RS 5
OpenSolaris builds do not contain the closed source tree.  Instead,
the developer downloads a closed binaries tree and unpacks it.
.B ON_CLOSED_BINS
tells nightly
where to find these closed binaries, so that it can add them into the
build.
.LP
.RE
.B ON_CRYPTO_BINS
.RS 5
This is the path to a compressed tarball that contains debug
cryptographic binaries that have been signed to allow execution
outside of Sun, e.g., $PARENT_WS/packages/$MACH/on-crypto.$MACH.bz2.
.I nightly
will automatically adjust the path for non-debug builds.  This tarball
is needed if the closed-source tree is not present.  Also, it is
usually needed when generating OpenSolaris deliverables from a project
workspace.  This is because most projects do not have access to the
necessary key and certificate that would let them sign their own
cryptographic binaries.
.LP
.RE
.B CHECK_PATHS
.RS 5
Normally, nightly runs the 'checkpaths' script to check for
discrepancies among the files that list paths to other files, such as
exception lists and req.flg.  Set this flag to 'n' to disable this
check, which appears in the nightly output as "Check lists of files."
.RE
.LP
.B CHECK_DMAKE
.RS 5
Nightly validates that the version of dmake encountered is known to be
safe to use.  Set this flag to 'n' to disable this test, allowing any
version of dmake to be used.
.RE
.LP
.B MULTI_PROTO
.RS 5
If "no" (the default),
.I nightly
will reuse $ROOT for both the DEBUG and non-DEBUG builds.  If "yes",
the DEBUG build will go in $ROOT and the non-DEBUG build will go in
$ROOT-nd.  Other values will be treated as "no".
$ROOT-nd.  Other values will be treated as "no".  Use of the
.B \-O
```



```
flag forces MULTI_PROTO to "yes".
.RE
.LP
.SH NIGHTLY HOOK ENVIRONMENT VARIABLES
.LP
Several optional environment variables may specify commands to run at
various points during the build.  Commands specified in the hook
variable will be run in a subshell; command output will be appended to
the mail message and log file.  If the hook exits with a non-zero
status, the build is aborted immediately.  Environment variables
defined in the environment file will be available.
.LP
.B SYS_PRE_NIGHTLY
.RS 5
Run just after the workspace lock is acquired.  This is reserved for
per-build-machine customizations and should be set only in /etc/nightly.conf
.RE
.LP
.B PRE_NIGHTLY
.RS 5
Run just after SYS_PRE_NIGHTLY.
.RE
.LP
.B PRE_BRINGOVER
.RS 5
Run just before bringover is started; not run if no bringover is done.
.RE
.LP
.B POST_BRINGOVER
.RS 5
Run just after bringover completes; not run if no bringover is done.
.RE
.LP
.B POST_NIGHTLY
.RS 5
Run after the build completes, with the return status of nightly - one
of "Completed", "Interrupted", or "Failed" - available in the
environment variable NIGHTLY_STATUS.
.RE
.LP
.B SYS_POST_NIGHTLY
.RS 5
This is reserved for per-build-machine customizations, and runs
immedately after POST_NIGHTLY.
.RE
.LP
.SH REALMODE ENVIRONMENT VARIABLES
.LP
The following environment variables referenced by
.I nightly
are only required when the -X option is used.
.LP
.RE
.B IA32_IHV_WS
.RS 5
Reference to the IHV workspace containing IHV driver binaries.
The IHV workspace must be fully built before starting the ON realmode build.
.LP
.RE
.B IA32_IHV_ROOT
.RS 5
Reference to the IHV workspace proto area.
The IHV workspace must be fully built before starting the ON realmode build.
.LP
.RE
.B IA32_IHV_PKGS
```

```
618 .RS 5
619 Reference to the IHV workspace packages.  If this is empty or the directory
620 is non-existent, then nightly will skip copying the packages.
621 .LP
622 .RE
623 .B IA32_IHV_BINARY_PKGS
624 .RS 5
625 Reference to binary-only IHV packages.  If this is empty or the directory
626 is non-existent, then nightly will skip copying the packages.
627 .LP
628 .RE
629 .B SPARC_RM_PKGARCHIVE
630 .RS 5
631 Destination for sparc realmode package SUNWrmodu.
632 Yes, this sparc package really is built on x86.
633 .SH FILES
634 .LP
635 .RS 5
636 /etc/nightly.conf
637 .RE
638 .LP
639 If present, nightly executes this file just prior to executing the
640 .I env
641 file.
642 .SH BUILDING THE IHV WORKSPACE
643 .LP
644 The IHV workspace can be built with
645 .I nightly.
646 The recommended options are:
647 .LP
648 .RS 5
649 NIGHTLY_OPTIONS="-pmWN"
650 .RE
651 .LP
652 None of the realmode environment variables needed for ON realmode builds
653 are required to build the IHV workspace.
654 .SH EXAMPLES
655 .LP
656 Start with the example file in usr/src/tools/env/developer.sh
657 (or gatekeeper.sh), copy to myenv and make your changes.
658 .LP
659 .PD 0
660 # grep NIGHTLY_OPTIONS myenv
661 .LP
662 NIGHTLY_OPTIONS="-ACrlapDm"
663 .LP
664 export NIGHTLY_OPTIONS
665 .LP
666 # /opt/onbld/bin/nightly -i myenv
667 .PD
668 .LP
669 .SH SEE ALSO
670 .BR bldenv (1)
```

```
**********************************************************
   70519 Wed Oct 16 17:40:06 2013
new/usr/src/tools/scripts/nightly.sh
4030 remove useless nightly/bldenv options
Reviewed by: Andy Stormont <andyjstormont@gmail.com>
**********************************************************
______unchanged_portion_omitted_

233 #
234 # Copy some or all of the source tree.
235 #
236 # Returns 0 for success, non-zero for failure.
237 #
238 # usage: copy_source CODEMGR_WS DESTDIR LABEL SRCROOT
239 #
240 function copy_source {
241         WS=$1
242         DEST=$2
243         label=$3
244         srcroot=$4

246         printf "\n==== Creating %s source from %s (%s) ====\n\n" \
247             "$DEST" "$WS" "$label" | tee -a $mail_msg_file >> $LOGFILE

249         printf "cleaning out %s\n" "$DEST." >> $LOGFILE
250         rm -rf "$DEST" >> $LOGFILE 2>&1

252         printf "creating %s\n" "$DEST." >> $LOGFILE
253         mkdir -p "$DEST" 2>> $LOGFILE

255         if (( $? != 0 )) ; then
256                 printf "failed to create %s\n" "$DEST" |
257                     tee -a $mail_msg_file >> $LOGFILE
258                 build_ok=n
259                 return 1
260         fi
261         cd "$WS"

263         printf "populating %s\n" "$DEST." >> $LOGFILE

265         case "$SCM_TYPE" in
266         teamware)
267                 find $srcroot -name 's\.*' -a -type f -print | \
268                     sed -e 's,SCCS\/s.,,' | \
269                     grep -v '/\.del-*' | \
270                     cpio -pd $DEST >>$LOGFILE 2>&1
271                 if (( $? != 0 )) ; then
272                     printf "cpio failed for %s\n" "$DEST" |
273                         tee -a $mail_msg_file >> $LOGFILE
274                     build_ok=n
275                     return 1
276                 fi
277                 ;;
278         mercurial)
279                 copy_source_mercurial $DEST $srcroot
280                 if (( $? != 0 )) ; then
281                     build_ok=n
282                     return 1
283                 fi
284                 ;;
285         *)
286                 build_ok=n
287                 echo "Tree copy is not supported for workspace type" \
288                     "$SCM_TYPE" | tee -a $mail_msg_file >> $LOGFILE
289                 return 1
290                 ;;
```



```
291         esac

293         return 0
294 }

296 #
297 # Mercurial-specific copy code for copy_source().
298 #
299 # Returns 0 for success, non-zero for failure.
300 #
301 # usage: copy_source_mercurial destdir srcroot
302 #
303 function copy_source_mercurial {
304         typeset dest=$1
305         typeset srcroot=$2

307         hg locate -I "$srcroot" | cpio -pd "$dest" >>$LOGFILE 2>&1
308         if (( $? != 0 )) ; then
309             printf "cpio failed for %s\n" "$dest" |
310                 tee -a $mail_msg_file >> $LOGFILE
311             return 1
312         fi

314         return 0
315 }

317 #
318 # function to create (but not build) the export/crypt source tree.
319 # usage: set_up_source_build CODEMGR_WS DESTDIR MAKE_TARGET
320 # Sets SRC to the modified source tree, for use by the caller when it
321 # builds the tree.
322 #
323 function set_up_source_build {
324         WS=$1
325         DEST=$2
326         MAKETARG=$3

328         copy_source $WS $DEST $MAKETARG usr
329         if (( $? != 0 )); then
330             echo "\nCould not copy source tree for source build." |
331                 tee -a $mail_msg_file >> $LOGFILE
332             build_ok=n
333             return
334         fi

336         SRC=${DEST}/usr/src

338         cd $SRC
339         rm -f ${MAKETARG}.out
340         echo "making ${MAKETARG} in ${SRC}." >> $LOGFILE
341         /bin/time $MAKE -e ${MAKETARG} 2>&1 | \
342             tee -a $SRC/${MAKETARG}.out >> $LOGFILE
343         echo "\n==== ${MAKETARG} build errors ====\n" >> $mail_msg_file
344         egrep ":" $SRC/${MAKETARG}.out | \
345                 egrep -e "(^${MAKE}:|[  ]error[:        \n])" | \
346                 egrep -v "Ignoring unknown host" | \
347                 egrep -v "warning" >> $mail_msg_file

349         echo "clearing state files." >> $LOGFILE
350         find . -name '.make*' -exec rm -f {} \;
351 }

233 # Return library search directive as function of given root.
234 function myldlibs {
235         echo "-L$1/lib -L$1/usr/lib"
236 }
______unchanged_portion_omitted_
```

```
243 #
244 # Function to do the build, including package generation.
245 # usage: build LABEL SUFFIX ND MULTIPROTO
246 # - LABEL is used to tag build output.
247 # - SUFFIX is used to distinguish files (e.g., DEBUG vs non-DEBUG,
248 #   open-only vs full tree).
249 # - ND is "-nd" (non-DEBUG builds) or "" (DEBUG builds).
250 # - If MULTIPROTO is "yes", it means to name the proto area according to
251 #   SUFFIX.  Otherwise ("no"), (re)use the standard proto area.
252 #
253 function build {
254         LABEL=$1
255         SUFFIX=$2
256         ND=$3
257         MULTIPROTO=$4
258         INSTALLOG=install${SUFFIX}-${MACH}
259         NOISE=noise${SUFFIX}-${MACH}
260         PKGARCHIVE=${PKGARCHIVE_ORIG}${SUFFIX}

262         ORIGROOT=$ROOT
263         [ $MULTIPROTO = no ] || export ROOT=$ROOT$SUFFIX

385         if [[ "$O_FLAG" = y ]]; then
386                 echo "\nSetting CLOSEDROOT= ${ROOT}-closed\n" >> $LOGFILE
387                 export CLOSEDROOT=${ROOT}-closed
388         fi

265         export ENVLDLIBS1='myldlibs $ROOT'
266         export ENVCPPFLAGS1='myheaders $ROOT'

268         this_build_ok=y
269         #
270         #       Build OS-Networking source
271         #
272         echo "\n==== Building OS-Net source at 'date' ($LABEL) ====\n" \
273                 >> $LOGFILE

275         rm -f $SRC/${INSTALLOG}.out
276         cd $SRC
277         /bin/time $MAKE -e install 2>&1 | \
278             tee -a $SRC/${INSTALLOG}.out >> $LOGFILE

280         if [[ "$SCM_TYPE" = teamware ]]; then
281                 echo "\n==== SCCS Noise ($LABEL) ====\n" >> $mail_msg_file
282                 egrep 'sccs(check:| *get)' $SRC/${INSTALLOG}.out >> \
283                         $mail_msg_file
284         fi

286         echo "\n==== Build errors ($LABEL) ====\n" >> $mail_msg_file
287         egrep ":" $SRC/${INSTALLOG}.out |
288                 egrep -e "(^${MAKE}:|[  ]error[:        \n])" | \
289                 egrep -v "Ignoring unknown host" | \
290                 egrep -v "cc .* -o error " | \
291                 egrep -v "warning" >> $mail_msg_file
292         if [ "$?" = "0" ]; then
293                 build_ok=n
294                 this_build_ok=n
295         fi
296         grep "bootblock image is .* bytes too big" $SRC/${INSTALLOG}.out \
297                 >> $mail_msg_file
298         if [ "$?" = "0" ]; then
299                 build_ok=n
300                 this_build_ok=n
301         fi
```



```
303         if [ "$W_FLAG" = "n" ]; then
304                 echo "\n==== Build warnings ($LABEL) ====\n" >>$mail_msg_file
305                 egrep -i warning: $SRC/${INSTALLOG}.out \
306                         | egrep -v '^tic:' \
307                         | egrep -v "symbol (\'|')timezone' has differing types:"
308                         | egrep -v "parameter <PSTAMP> set to" \
309                         | egrep -v "Ignoring unknown host" \
310                         | egrep -v "redefining segment flags attribute for" \
311                         >> $mail_msg_file
312         fi

314         echo "\n==== Ended OS-Net source build at 'date' ($LABEL) ====\n" \
315                 >> $LOGFILE

317         echo "\n==== Elapsed build time ($LABEL) ====\n" >>$mail_msg_file
318         tail -3  $SRC/${INSTALLOG}.out >>$mail_msg_file

320         if [ "$i_FLAG" = "n" -a "$W_FLAG" = "n" ]; then
321                 rm -f $SRC/${NOISE}.ref
322                 if [ -f $SRC/${NOISE}.out ]; then
323                         mv $SRC/${NOISE}.out $SRC/${NOISE}.ref
324                 fi
325                 grep : $SRC/${INSTALLOG}.out \
326                         | egrep -v '^/' \
327                         | egrep -v '^(Start|Finish|real|user|sys|./bld_awk)' \
328                         | egrep -v '^tic:' \
329                         | egrep -v '^mcs' \
330                         | egrep -v '^LD_LIBRARY_PATH=' \
331                         | egrep -v 'ar: creating' \
332                         | egrep -v 'ar: writing' \
333                         | egrep -v 'conflicts:' \
334                         | egrep -v ':saved created' \
335                         | egrep -v '^stty.*c:' \
336                         | egrep -v '^mfgname.c:' \
337                         | egrep -v '^uname-i.c:' \
338                         | egrep -v '^volumes.c:' \
339                         | egrep -v '^lint library construction:' \
340                         | egrep -v 'tsort: INFORM:' \
341                         | egrep -v 'stripalign:' \
342                         | egrep -v 'chars, width' \
343                         | egrep -v "symbol (\'|')timezone' has differing types:"
344                         | egrep -v 'PSTAMP' \
345                         | egrep -v '|%WHOANDWHERE%|' \
346                         | egrep -v '^Manifying' \
347                         | egrep -v 'Ignoring unknown host' \
348                         | egrep -v 'Processing method:' \
349                         | egrep -v '^Writing' \
350                         | egrep -v 'spellin1:' \
351                         | egrep -v '^adding:' \
352                         | egrep -v "^echo 'msgid" \
353                         | egrep -v '^echo ' \
354                         | egrep -v '\.c:$' \
355                         | egrep -v '^Adding file:' \
356                         | egrep -v 'CLASSPATH=' \
357                         | egrep -v '\/var\/mail\/:saved' \
358                         | egrep -v -- '-DUTS_VERSION=' \
359                         | egrep -v '^Running Mkbootstrap' \
360                         | egrep -v '^Applet length read:' \
361                         | egrep -v 'bytes written:' \
362                         | egrep -v '^File:SolarisAuthApplet.bin' \
363                         | egrep -v -i 'jibversion' \
364                         | egrep -v '^Output size:' \
365                         | egrep -v '^Solo size statistics:' \
366                         | egrep -v '^Using ROM API Version' \
367                         | egrep -v '^Zero Signature length:' \
368                         | egrep -v '^Note \(probably harmless\):' \
```

```
369                         | egrep -v '::' \
370                         | egrep -v -- '-xcache' \
371                         | egrep -v '^\+' \
372                         | egrep -v '^cc1: note: -fwritable-strings' \
373                         | egrep -v 'svccfg-native -s svc:/' \
374                         | sort | uniq >$SRC/${NOISE}.out
375                 if [ ! -f $SRC/${NOISE}.ref ]; then
376                         cp $SRC/${NOISE}.out $SRC/${NOISE}.ref
377                 fi
378                 echo "\n==== Build noise differences ($LABEL) ====\n" \
379                         >>$mail_msg_file
380                 diff $SRC/${NOISE}.ref $SRC/${NOISE}.out >>$mail_msg_file
381         fi

383         #
384         #       Re-sign selected binaries using signing server
385         #       (gatekeeper builds only)
386         #
387         if [ -n "$CODESIGN_USER" -a "$this_build_ok" = "y" ]; then
388                 echo "\n==== Signing proto area at 'date' ====\n" >> $LOGFILE
389                 signing_file="${TMPDIR}/signing"
390                 rm -f ${signing_file}
391                 export CODESIGN_USER
392                 signproto $SRC/tools/codesign/creds 2>&1 | \
393                         tee -a ${signing_file} >> $LOGFILE
394                 echo "\n==== Finished signing proto area at 'date' ====\n" \
395                     >> $LOGFILE
396                 echo "\n==== Crypto module signing errors ($LABEL) ====\n" \
397                     >> $mail_msg_file
398                 egrep 'WARNING|ERROR' ${signing_file} >> $mail_msg_file
399                 if (( $? == 0 )) ; then
400                         build_ok=n
401                         this_build_ok=n
402                 fi
403         fi

405         #
406         #       Building Packages
407         #
408         if [ "$p_FLAG" = "y" -a "$this_build_ok" = "y" ]; then
409                 if [ -d $SRC/pkg -o -d $SRC/pkgdefs ]; then
410                         echo "\n==== Creating $LABEL packages at 'date' ====\n"
411                                 >> $LOGFILE
412                         echo "Clearing out $PKGARCHIVE ..." >> $LOGFILE
413                         rm -rf $PKGARCHIVE >> "$LOGFILE" 2>&1
414                         mkdir -p $PKGARCHIVE >> "$LOGFILE" 2>&1

416                         for d in pkg pkgdefs; do
417                                 if [ ! -f "$SRC/$d/Makefile" ]; then
418                                         continue
419                                 fi
420                                 rm -f $SRC/$d/${INSTALLOG}.out
421                                 cd $SRC/$d
422                                 /bin/time $MAKE -e install 2>&1 | \
423                                         tee -a $SRC/$d/${INSTALLOG}.out >> $LOGF
424                         done

426                         echo "\n==== package build errors ($LABEL) ====\n" \
427                                 >> $mail_msg_file

429                         for d in pkg pkgdefs; do
430                                 if [ ! -f "$SRC/$d/Makefile" ]; then
431                                         continue
432                                 fi

434                                 egrep "${MAKE}|ERROR|WARNING" $SRC/$d/${INSTALLO
```



```
435                                         grep ':' | \
436                                         grep -v PSTAMP | \
437                                         egrep -v "Ignoring unknown host" \
438                                         >> $mail_msg_file
439                         done
440                 else
441                         #
442                         # Handle it gracefully if -p was set but there are
443                         # neither pkg nor pkgdefs directories.
444                         #
445                         echo "\n==== No $LABEL packages to build ====\n" \
446                                 >> $LOGFILE
447                 fi
448         else
449                 echo "\n==== Not creating $LABEL packages ====\n" >> $LOGFILE
450         fi

452         ROOT=$ORIGROOT
453 }
```

_____***unchanged_portion_omitted_***

```
776 MACH='uname -p'

778 if [ "$OPTHOME" = "" ]; then
779         OPTHOME=/opt
780         export OPTHOME
781 fi
782 if [ "$TEAMWARE" = "" ]; then
783         TEAMWARE=$OPTHOME/teamware
784         export TEAMWARE
785 fi

787 USAGE='Usage: nightly [-in] [+t] [-V VERS ] <env_file>
912 USAGE='Usage: nightly [-in] [+t] [-V VERS ] [ -S E|D|H|O ] <env_file>

789 Where:
790         -i      Fast incremental options (no clobber, lint, check)
791         -n      Do not do a bringover
792         +t      Use the build tools in $ONBLD_TOOLS/bin
793         -V VERS set the build version string to VERS
919         -S      Build a variant of the source product
920                 E - build exportable source
921                 D - build domestic source (exportable + crypt)
922                 H - build hybrid source (binaries + deleted source)
923                 O - build (only) open source

795         <env_file>  file in Bourne shell syntax that sets and exports
796         variables that configure the operation of this script and many of
797         the scripts this one calls. If <env_file> does not exist,
798         it will be looked for in $OPTHOME/onbld/env.

800 non-DEBUG is the default build type. Build options can be set in the
801 NIGHTLY_OPTIONS variable in the <env_file> as follows:

803         -A      check for ABI differences in .so files
804         -C      check for cstyle/hdrchk errors
805         -D      do a build with DEBUG on
806         -F      do _not_ do a non-DEBUG build
807         -G      gate keeper default group of options (-au)
808         -I      integration engineer default group of options (-ampu)
809         -M      do not run pmodes (safe file permission checker)
810         -N      do not run protocmp
941         -O      generate OpenSolaris deliverables
811         -R      default group of options for building a release (-mp)
812         -U      update proto area in the parent
```

```
 813         -V VERS set the build version string to VERS
 814         -X      copy x86 IHV proto area
 815         -f      find unreferenced files
 816         -i      do an incremental build (no "make clobber")
 817         -l      do "make lint" in $LINTDIRS (default: $SRC y)
 818         -m      send mail to $MAILTO at end of build
 819         -n      do not do a bringover
 820         -o      build using root privileges to set OWNER/GROUP (old style)
 821         -p      create packages
 822         -r      check ELF runtime attributes in the proto area
 823         -t      build and use the tools in $SRC/tools (default setting)
 824         +t      Use the build tools in $ONBLD_TOOLS/bin
 825         -u      update proto_list_$MACH and friends in the parent workspace;
 826                 when used with -f, also build an unrefmaster.out in the parent
 827         -w      report on differences between previous and current proto areas
 828         -z      compress cpio archives with gzip
 829         -W      Do not report warnings (freeware gate ONLY)
 961         -S      Build a variant of the source product
 962                 E - build exportable source
 963                 D - build domestic source (exportable + crypt)
 964                 H - build hybrid source (binaries + deleted source)
 965                 O - build (only) open source
 830 '
 831 #
 832 #       A log file will be generated under the name $LOGFILE
 833 #       for partially completed build and log.`date '+%F'`
 834 #       in the same directory for fully completed builds.
 835 #

 837 # default values for low-level FLAGS; G I R are group FLAGS
 838 A_FLAG=n
 839 C_FLAG=n
 840 D_FLAG=n
 841 F_FLAG=n
 842 f_FLAG=n
 843 i_FLAG=n; i_CMD_LINE_FLAG=n
 844 l_FLAG=n
 845 M_FLAG=n
 846 m_FLAG=n
 847 N_FLAG=n
 848 n_FLAG=n
 985 O_FLAG=n
 849 o_FLAG=n
 850 P_FLAG=n
 851 p_FLAG=n
 852 r_FLAG=n
 853 T_FLAG=n
 854 t_FLAG=y
 855 U_FLAG=n
 856 u_FLAG=n
 857 V_FLAG=n
 858 W_FLAG=n
 859 w_FLAG=n
 860 X_FLAG=n
 998 SD_FLAG=n
 999 SE_FLAG=n
1000 SH_FLAG=n
1001 SO_FLAG=n
 861 #
 862 XMOD_OPT=
 863 #
 864 build_ok=y

1007 function is_source_build {
1008         [ "$SE_FLAG" = "y" -o "$SD_FLAG" = "y" -o \
1009             "$SH_FLAG" = "y" -o "$SO_FLAG" = "y" ]
```



```
1010         return $?
1011 }

 866 #
 867 # examine arguments
 868 #

1017 #
1018 # single function for setting -S flag and doing error checking.
1019 # usage: set_S_flag <type>
1020 # where <type> is the source build type ("E", "D", ...).
1021 #
1022 function set_S_flag {
1023         if is_source_build; then
1024                 echo "Can only build one source variant at a time."
1025                 exit 1
1026         fi
1027         if [ "$1" = "E" ]; then
1028                 SE_FLAG=y
1029         elif [ "$1" = "D" ]; then
1030                 SD_FLAG=y
1031         elif [ "$1" = "H" ]; then
1032                 SH_FLAG=y
1033         elif [ "$1" = "O" ]; then
1034                 SO_FLAG=y
1035         else
1036                 echo "$USAGE"
1037                 exit 1
1038         fi
1039 }

 870 OPTIND=1
 871 while getopts +intV: FLAG
1042 while getopts +inS:tV: FLAG
 872 do
 873         case $FLAG in
 874           i )   i_FLAG=y; i_CMD_LINE_FLAG=y
 875                 ;;
 876           n )   n_FLAG=y
 877                 ;;
1049           S )
1050                 set_S_flag $OPTARG
1051                 ;;
 878          +t )   t_FLAG=n
 879                 ;;
 880           V )   V_FLAG=y
 881                 V_ARG="$OPTARG"
 882                 ;;
 883          \? )   echo "$USAGE"
 884                 exit 1
 885                 ;;
 886         esac
 887 done

 889 # correct argument count after options
 890 shift `expr $OPTIND - 1`

 892 # test that the path to the environment-setting file was given
 893 if [ $# -ne 1 ]; then
 894         echo "$USAGE"
 895         exit 1
 896 fi

 898 # check if user is running nightly as root
 899 # ISUSER is set non-zero if an ordinary user runs nightly, or is zero
 900 # when root invokes nightly.
```

```
/usr/bin/id | grep '^uid=0(' >/dev/null 2>&1
ISUSER=$?;      export ISUSER

#
# force locale to C
LC_COLLATE=C;   export LC_COLLATE
LC_CTYPE=C;     export LC_CTYPE
LC_MESSAGES=C;  export LC_MESSAGES
LC_MONETARY=C;  export LC_MONETARY
LC_NUMERIC=C;   export LC_NUMERIC
LC_TIME=C;      export LC_TIME

# clear environment variables we know to be bad for the build
unset LD_OPTIONS
unset LD_AUDIT          LD_AUDIT_32             LD_AUDIT_64
unset LD_BIND_NOW       LD_BIND_NOW_32          LD_BIND_NOW_64
unset LD_BREADTH        LD_BREADTH_32           LD_BREADTH_64
unset LD_CONFIG         LD_CONFIG_32            LD_CONFIG_64
unset LD_DEBUG          LD_DEBUG_32             LD_DEBUG_64
unset LD_DEMANGLE       LD_DEMANGLE_32          LD_DEMANGLE_64
unset LD_FLAGS          LD_FLAGS_32             LD_FLAGS_64
unset LD_LIBRARY_PATH   LD_LIBRARY_PATH_32      LD_LIBRARY_PATH_64
unset LD_LOADFLTR       LD_LOADFLTR_32          LD_LOADFLTR_64
unset LD_NOAUDIT        LD_NOAUDIT_32           LD_NOAUDIT_64
unset LD_NOAUXFLTR      LD_NOAUXFLTR_32         LD_NOAUXFLTR_64
unset LD_NOCONFIG       LD_NOCONFIG_32          LD_NOCONFIG_64
unset LD_NODIRCONFIG    LD_NODIRCONFIG_32       LD_NODIRCONFIG_64
unset LD_NODIRECT       LD_NODIRECT_32          LD_NODIRECT_64
unset LD_NOLAZYLOAD     LD_NOLAZYLOAD_32        LD_NOLAZYLOAD_64
unset LD_NOOBJALTER     LD_NOOBJALTER_32        LD_NOOBJALTER_64
unset LD_NOVERSION      LD_NOVERSION_32         LD_NOVERSION_64
unset LD_ORIGIN         LD_ORIGIN_32            LD_ORIGIN_64
unset LD_PRELOAD        LD_PRELOAD_32           LD_PRELOAD_64
unset LD_PROFILE        LD_PROFILE_32           LD_PROFILE_64

unset CONFIG
unset GROUP
unset OWNER
unset REMOTE
unset ENV
unset ARCH
unset CLASSPATH
unset NAME

#
# To get ONBLD_TOOLS from the environment, it must come from the env file.
# If it comes interactively, it is generally TOOLS_PROTO, which will be
# clobbered before the compiler version checks, which will therefore fail.
#
unset ONBLD_TOOLS

#
#       Setup environmental variables
#
if [ -f /etc/nightly.conf ]; then
        . /etc/nightly.conf
fi

if [ -f $1 ]; then
        if [[ $1 = */* ]]; then
                . $1
        else
                . ./$1
        fi
else
        if [ -f $OPTHOME/onbld/env/$1 ]; then
```



```
                . $OPTHOME/onbld/env/$1
        else
                echo "Cannot find env file as either $1 or $OPTHOME/onbld/env/$1
                exit 1
        fi
fi

# contents of stdenv.sh inserted after next line:
# STDENV_START
# STDENV_END

# Check if we have sufficient data to continue...
[[ -v CODEMGR_WS ]] || fatal_error "Error: Variable CODEMGR_WS not set."
if  [[ "${NIGHTLY_OPTIONS}" == ~(F)n ]] ; then
        # Check if the gate data are valid if we don't do a "bringover" below
        [[ -d "${CODEMGR_WS}" ]] || \
                fatal_error "Error: ${CODEMGR_WS} is not a directory."
        [[ -f "${CODEMGR_WS}/usr/src/Makefile" ]] || \
                fatal_error "Error: ${CODEMGR_WS}/usr/src/Makefile not found."
fi

#
# place ourselves in a new task, respecting BUILD_PROJECT if set.
#
if [ -z "$BUILD_PROJECT" ]; then
        /usr/bin/newtask -c $$
else
        /usr/bin/newtask -c $$ -p $BUILD_PROJECT
fi

ps -o taskid= -p $$ | read build_taskid
ps -o project= -p $$ | read build_project

#
# See if NIGHTLY_OPTIONS is set
#
if [ "$NIGHTLY_OPTIONS" = "" ]; then
        NIGHTLY_OPTIONS="-aBm"
fi

#
# If BRINGOVER_WS was not specified, let it default to CLONE_WS
#
if [ "$BRINGOVER_WS" = "" ]; then
        BRINGOVER_WS=$CLONE_WS
fi

#
# If CLOSED_BRINGOVER_WS was not specified, let it default to CLOSED_CLONE_WS
#
if [ "$CLOSED_BRINGOVER_WS" = "" ]; then
        CLOSED_BRINGOVER_WS=$CLOSED_CLONE_WS
fi

#
# If BRINGOVER_FILES was not specified, default to usr
#
if [ "$BRINGOVER_FILES" = "" ]; then
        BRINGOVER_FILES="usr"
fi

check_closed_tree

#
# Note: changes to the option letters here should also be applied to the
#       bldenv script.  'd' is listed for backward compatibility.
```

```
1033 #
1034 NIGHTLY_OPTIONS=-${NIGHTLY_OPTIONS#-}
1035 OPTIND=1
1036 while getopts +ABCDdFfGIilMmNnoPpRrTtUuWwXxz FLAG $NIGHTLY_OPTIONS
1210 while getopts +ABCDdFfGIilMmNnOoPpRrS:TtUuWwXxz FLAG $NIGHTLY_OPTIONS
1037 do
1038         case $FLAG in
1039           A )   A_FLAG=y
1040                 ;;
1041           B )   D_FLAG=y
1042                 ;; # old version of D
1043           C )   C_FLAG=y
1044                 ;;
1045           D )   D_FLAG=y
1046                 ;;
1047           F )   F_FLAG=y
1048                 ;;
1049           f )   f_FLAG=y
1050                 ;;
1051           G )   u_FLAG=y
1052                 ;;
1053           I )   m_FLAG=y
1054                 p_FLAG=y
1055                 u_FLAG=y
1056                 ;;
1057           i )   i_FLAG=y
1058                 ;;
1059           l )   l_FLAG=y
1060                 ;;
1061           M )   M_FLAG=y
1062                 ;;
1063           m )   m_FLAG=y
1064                 ;;
1065           N )   N_FLAG=y
1066                 ;;
1067           n )   n_FLAG=y
1068                 ;;
1243           O )   O_FLAG=y
1244                 ;;
1069           o )   o_FLAG=y
1070                 ;;
1071           P )   P_FLAG=y
1072                 ;; # obsolete
1073           p )   p_FLAG=y
1074                 ;;
1075           R )   m_FLAG=y
1076                 p_FLAG=y
1077                 ;;
1078           r )   r_FLAG=y
1079                 ;;
1256           S )
1257                 set_S_flag $OPTARG
1258                 ;;
1080           T )   T_FLAG=y
1081                 ;; # obsolete
1082          +t )   t_FLAG=n
1083                 ;;
1084           U )   if [ -z "${PARENT_ROOT}" ]; then
1085                         echo "PARENT_ROOT must be set if the U flag is" \
1086                             "present in NIGHTLY_OPTIONS."
1087                         exit 1
1088                 fi
1089                 NIGHTLY_PARENT_ROOT=$PARENT_ROOT
1090                 if [ -n "${PARENT_TOOLS_ROOT}" ]; then
1091                         NIGHTLY_PARENT_TOOLS_ROOT=$PARENT_TOOLS_ROOT
1092                 fi
```



```
1093                 U_FLAG=y
1094                 ;;
1095           u )   u_FLAG=y
1096                 ;;
1097           W )   W_FLAG=y
1098                 ;;
1100           w )   w_FLAG=y
1101                 ;;
1102           X )   # now that we no longer need realmode builds, just
1103                 # copy IHV packages.  only meaningful on x86.
1104                 if [ "$MACH" = "i386" ]; then
1105                         X_FLAG=y
1106                 fi
1107                 ;;
1108           x )   XMOD_OPT="-x"
1109                 ;;
1110          \? )   echo "$USAGE"
1111                 exit 1
1112                 ;;
1113         esac
1114 done

1116 if [ $ISUSER -ne 0 ]; then
1117         if [ "$o_FLAG" = "y" ]; then
1118                 echo "Old-style build requires root permission."
1119                 exit 1
1120         fi

1122         # Set default value for STAFFER, if needed.
1123         if [ -z "$STAFFER" -o "$STAFFER" = "nobody" ]; then
1124                 STAFFER=`/usr/xpg4/bin/id -un`
1125                 export STAFFER
1126         fi
1127 fi

1129 if [ -z "$MAILTO" -o "$MAILTO" = "nobody" ]; then
1130         MAILTO=$STAFFER
1131         export MAILTO
1132 fi

1134 PATH="$OPTHOME/onbld/bin:$OPTHOME/onbld/bin/${MACH}:/usr/ccs/bin"
1135 PATH="$PATH:$OPTHOME/SUNWspro/bin:$TEAMWARE/bin:/usr/bin:/usr/sbin:/usr/ucb"
1136 PATH="$PATH:/usr/openwin/bin:/usr/sfw/bin:/opt/sfw/bin:."
1137 export PATH

1139 # roots of source trees, both relative to $SRC and absolute.
1140 relsrcdirs="."
1141 abssrcdirs="$SRC"

1143 unset CH
1144 if [ "$o_FLAG" = "y" ]; then
1145 # root invoked old-style build -- make sure it works as it always has
1146 # by exporting 'CH'.  The current Makefile.master doesn't use this, but
1147 # the old ones still do.
1148         PROTOCMPTERSE="protocmp.terse"
1149         CH=
1150         export CH
1151 else
1152         PROTOCMPTERSE="protocmp.terse -gu"
1153 fi
1154 POUND_SIGN="#"
1155 # have we set RELEASE_DATE in our env file?
1156 if [ -z "$RELEASE_DATE" ]; then
1157         RELEASE_DATE=$(LC_ALL=C date +"%B %Y")
1158 fi
```

```
1159 BUILD_DATE=$(LC_ALL=C date +%Y-%b-%d)
1160 BASEWSDIR=$(basename $CODEMGR_WS)
1161 DEV_CM="\"@(#)SunOS Internal Development: $LOGNAME $BUILD_DATE [$BASEWSDIR]\""

1163 # we export POUND_SIGN, RELEASE_DATE and DEV_CM to speed up the build process
1164 # by avoiding repeated shell invocations to evaluate Makefile.master definitions
1165 # we export o_FLAG and X_FLAG for use by makebfu, and by usr/src/pkg/Makefile
1166 export o_FLAG X_FLAG POUND_SIGN RELEASE_DATE DEV_CM

1168 maketype="distributed"
1169 MAKE=dmake
1170 # get the dmake version string alone
1171 DMAKE_VERSION=$( $MAKE -v )
1172 DMAKE_VERSION=${DMAKE_VERSION#*: }
1173 # focus in on just the dotted version number alone
1174 DMAKE_MAJOR=$( echo $DMAKE_VERSION | \
1175         sed -e 's/.*\<\([^.]*\.[^   ]*\).*$/\1/' )
1176 # extract the second (or final) integer
1177 DMAKE_MINOR=${DMAKE_MAJOR#*.}
1178 DMAKE_MINOR=${DMAKE_MINOR%%.*}
1179 # extract the first integer
1180 DMAKE_MAJOR=${DMAKE_MAJOR%%.*}
1181 CHECK_DMAKE=${CHECK_DMAKE:-y}
1182 # x86 was built on the 12th, sparc on the 13th.
1183 if [ "$CHECK_DMAKE" = "y" -a \
1184      "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/12" -a \
1185      "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/13" -a \( \
1186      "$DMAKE_MAJOR" -lt 7 -o \
1187      "$DMAKE_MAJOR" -eq 7 -a "$DMAKE_MINOR" -lt 4 \) ]; then
1188         if [ -z "$DMAKE_VERSION" ]; then
1189                 echo "$MAKE is missing."
1190                 exit 1
1191         fi
1192         echo 'whence $MAKE'" version is:"
1193         echo "  ${DMAKE_VERSION}"
1194         cat <<EOF

1196 This version may not be safe for use.  Either set TEAMWARE to a better
1197 path or (if you really want to use this version of dmake anyway), add
1198 the following to your environment to disable this check:

1200   CHECK_DMAKE=n
1201 EOF
1202         exit 1
1203 fi
1204 export PATH
1205 export MAKE

1207 if [ "${SUNWSPRO}" != "" ]; then
1208         PATH="${SUNWSPRO}/bin:$PATH"
1209         export PATH
1210 fi

1212 hostname=$(uname -n)
1213 if [[ $DMAKE_MAX_JOBS != +([0-9]) || $DMAKE_MAX_JOBS -eq 0 ]]
1214 then
1215         maxjobs=
1216         if [[ -f $HOME/.make.machines ]]
1217         then
1218                 # Note: there is a hard tab and space character in the []s
1219                 # below.
1220                 egrep -i "^[    ]*$hostname[    \.]" \
1221                         $HOME/.make.machines | read host jobs
1222                 maxjobs=${jobs##*=}
1223         fi
```



```
1225         if [[ $maxjobs != +([0-9]) || $maxjobs -eq 0 ]]
1226         then
1227                 # default
1228                 maxjobs=4
1229         fi

1231         export DMAKE_MAX_JOBS=$maxjobs
1232 fi

1234 DMAKE_MODE=parallel;
1235 export DMAKE_MODE

1237 if [ -z "${ROOT}" ]; then
1238         echo "ROOT must be set."
1239         exit 1
1240 fi

1242 #
1243 # if -V flag was given, reset VERSION to V_ARG
1244 #
1245 if [ "$V_FLAG" = "y" ]; then
1246         VERSION=$V_ARG
1247 fi

1249 #
1250 # Check for IHV root for copying ihv proto area
1251 #
1252 if [ "$X_FLAG" = "y" ]; then
1253         if [ "$IA32_IHV_ROOT" = "" ]; then
1254                 echo "IA32_IHV_ROOT: must be set for copying ihv proto"
1255                 args_ok=n
1256         fi
1257         if [ ! -d "$IA32_IHV_ROOT" ]; then
1258                 echo "$IA32_IHV_ROOT: not found"
1259                 args_ok=n
1260         fi
1261         if [ "$IA32_IHV_WS" = "" ]; then
1262                 echo "IA32_IHV_WS: must be set for copying ihv proto"
1263                 args_ok=n
1264         fi
1265         if [ ! -d "$IA32_IHV_WS" ]; then
1266                 echo "$IA32_IHV_WS: not found"
1267                 args_ok=n
1268         fi
1269 fi

1450 # Append source version
1451 if [ "$SE_FLAG" = "y" ]; then
1452         VERSION="${VERSION}:EXPORT"
1453 fi

1455 if [ "$SD_FLAG" = "y" ]; then
1456         VERSION="${VERSION}:DOMESTIC"
1457 fi

1459 if [ "$SH_FLAG" = "y" ]; then
1460         VERSION="${VERSION}:MODIFIED_SOURCE_PRODUCT"
1461 fi

1463 if [ "$SO_FLAG" = "y" ]; then
1464         VERSION="${VERSION}:OPEN_ONLY"
1465 fi

1271 TMPDIR="/tmp/nightly.tmpdir.$$"
1272 export TMPDIR
1273 rm -rf ${TMPDIR}
```

```
1274 mkdir -p $TMPDIR || exit 1
1275 chmod 777 $TMPDIR

1277 #
1278 # Keep elfsign's use of pkcs11_softtoken from looking in the user home
1279 # directory, which doesn't always work.   Needed until all build machines
1280 # have the fix for 6271754
1281 #
1282 SOFTTOKEN_DIR=$TMPDIR
1283 export SOFTTOKEN_DIR

1285 #
1286 # Tools should only be built non-DEBUG.  Keep track of the tools proto
1287 # area path relative to $TOOLS, because the latter changes in an
1288 # export build.
1289 #
1290 # TOOLS_PROTO is included below for builds other than usr/src/tools
1291 # that look for this location.  For usr/src/tools, this will be
1292 # overridden on the $MAKE command line in build_tools().
1293 #
1294 TOOLS=${SRC}/tools
1295 TOOLS_PROTO_REL=proto/root_${MACH}-nd
1296 TOOLS_PROTO=${TOOLS}/${TOOLS_PROTO_REL};        export TOOLS_PROTO

1298 unset   CFLAGS LD_LIBRARY_PATH LDFLAGS

1300 # create directories that are automatically removed if the nightly script
1301 # fails to start correctly
1302 function newdir {
1303         dir=$1
1304         toadd=
1305         while [ ! -d $dir ]; do
1306                 toadd="$dir $toadd"
1307                 dir=`dirname $dir`
1308         done
1309         torm=
1310         newlist=
1311         for dir in $toadd; do
1312                 if staffer mkdir $dir; then
1313                         newlist="$ISUSER $dir $newlist"
1314                         torm="$dir $torm"
1315                 else
1316                         [ -z "$torm" ] || staffer rmdir $torm
1317                         return 1
1318                 fi
1319         done
1320         newdirlist="$newlist $newdirlist"
1321         return 0
1322 }
```

*_____unchanged_portion_omitted_*

```
1495 #
1496 # Return the list of interesting proto areas, depending on the current
1497 # options.
1498 #
1499 function allprotos {
1500         typeset roots="$ROOT"

1502         if [[ "$F_FLAG" = n && "$MULTI_PROTO" = yes ]]; then
1503                 roots="$roots $ROOT-nd"
1504         fi

1702         if [[ $O_FLAG = y ]]; then
1703                 roots="$roots $ROOT-closed"
1704                 [ $MULTI_PROTO = yes ] && roots="$roots $ROOT-nd-closed"
1705         fi
```



```
1506         echo $roots
1507 }

1509 # Ensure no other instance of this script is running on this host.
1510 # LOCKNAME can be set in <env_file>, and is by default, but is not
1511 # required due to the use of $ATLOG below.
1512 if [ -n "$LOCKNAME" ]; then
1513         create_lock /tmp/$LOCKNAME "lockfile"
1514 fi
1515 #
1516 # Create from one, two, or three other locks:
1517 #       $ATLOG/nightly.lock
1518 #               - protects against multiple builds in same workspace
1519 #       $PARENT_WS/usr/src/nightly.$MACH.lock
1520 #               - protects against multiple 'u' copy-backs
1521 #       $NIGHTLY_PARENT_ROOT/nightly.lock
1522 #               - protects against multiple 'U' copy-backs
1523 #
1524 # Overriding ISUSER to 1 causes the lock to be created as root if the
1525 # script is run as root.  The default is to create it as $STAFFER.
1526 ISUSER=1 create_lock $ATLOG/nightly.lock "atloglockfile"
1527 if [ "$u_FLAG" = "y" ]; then
1528         create_lock $PARENT_WS/usr/src/nightly.$MACH.lock "ulockfile"
1529 fi
1530 if [ "$U_FLAG" = "y" ]; then
1531         # NIGHTLY_PARENT_ROOT is written as root if script invoked as root.
1532         ISUSER=1 create_lock $NIGHTLY_PARENT_ROOT/nightly.lock "Ulockfile"
1533 fi

1535 # Locks have been taken, so we're doing a build and we're committed to
1536 # the directories we may have created so far.
1537 newdirlist=

1539 #
1540 # Create mail_msg_file
1541 #
1542 mail_msg_file="${TMPDIR}/mail_msg"
1543 touch $mail_msg_file
1544 build_time_file="${TMPDIR}/build_time"
1545 build_environ_file="${TMPDIR}/build_environ"
1546 touch $build_environ_file
1547 #
1548 #       Move old LOGFILE aside
1549 #       ATLOG directory already made by 'create_lock' above
1550 #
1551 if [ -f $LOGFILE ]; then
1552         mv -f $LOGFILE ${LOGFILE}-
1553 fi
1554 #
1555 #       Build OsNet source
1556 #
1557 START_DATE=`date`
1558 SECONDS=0
1559 echo "\n==== Nightly $maketype build started:   $START_DATE ====" \
1560     | tee -a $LOGFILE > $build_time_file

1562 echo "\nBuild project:  $build_project\nBuild taskid:   $build_taskid" | \
1563     tee -a $mail_msg_file >> $LOGFILE

1565 # make sure we log only to the nightly build file
1566 build_noise_file="${TMPDIR}/build_noise"
1567 exec </dev/null >$build_noise_file 2>&1

1569 run_hook SYS_PRE_NIGHTLY
1570 run_hook PRE_NIGHTLY
```

```
echo "\n==== list of environment variables ====\n" >> $LOGFILE
env >> $LOGFILE

echo "\n==== Nightly argument issues ====\n" | tee -a $mail_msg_file >> $LOGFILE

if [ "$P_FLAG" = "y" ]; then
        obsolete_build GPROF | tee -a $mail_msg_file >> $LOGFILE
fi

if [ "$T_FLAG" = "y" ]; then
        obsolete_build TRACE | tee -a $mail_msg_file >> $LOGFILE
fi

if [ "$N_FLAG" = "y" ]; then
if is_source_build; then
        if [ "$i_FLAG" = "y" -o "$i_CMD_LINE_FLAG" = "y" ]; then
                echo "WARNING: the -S flags do not support incremental" \
                    "builds; forcing clobber\n" | tee -a $mail_msg_file >> $LOGF
                i_FLAG=n
                i_CMD_LINE_FLAG=n
        fi
        if [ "$N_FLAG" = "n" ]; then
                echo "WARNING: the -S flags do not support protocmp;" \
                    "protocmp disabled\n" | \
                    tee -a $mail_msg_file >> $LOGFILE
                N_FLAG=y
        fi
        if [ "$l_FLAG" = "y" ]; then
                echo "WARNING: the -S flags do not support lint;" \
                    "lint disabled\n" | tee -a $mail_msg_file >> $LOGFILE
                l_FLAG=n
        fi
        if [ "$C_FLAG" = "y" ]; then
                echo "WARNING: the -S flags do not support cstyle;" \
                    "cstyle check disabled\n" | tee -a $mail_msg_file >> $LOGFIL
                C_FLAG=n
        fi
else
        if [ "$N_FLAG" = "y" ]; then
        if [ "$p_FLAG" = "y" ]; then
                cat <<EOF | tee -a $mail_msg_file >> $LOGFILE
WARNING: the p option (create packages) is set, but so is the N option (do
         not run protocmp); this is dangerous; you should unset the N option
EOF
        else
                cat <<EOF | tee -a $mail_msg_file >> $LOGFILE
Warning: the N option (do not run protocmp) is set; it probably shouldn't be
EOF
        fi
        echo "" | tee -a $mail_msg_file >> $LOGFILE
        fi
fi

if [ "$D_FLAG" = "n" -a "$l_FLAG" = "y" ]; then
        #
        # In the past we just complained but went ahead with the lint
        # pass, even though the proto area was built non-DEBUG.  It's
        # unlikely that non-DEBUG headers will make a difference, but
        # rather than assuming it's a safe combination, force the user
        # to specify a DEBUG build.
        #
        echo "WARNING: DEBUG build not requested; disabling lint.\n" \
            | tee -a $mail_msg_file >> $LOGFILE
        l_FLAG=n
fi
```



```
if [ "$f_FLAG" = "y" ]; then
        if [ "$i_FLAG" = "y" ]; then
                echo "WARNING: the -f flag cannot be used during incremental" \
                    "builds; ignoring -f\n" | tee -a $mail_msg_file >> $LOGFILE
                f_FLAG=n
        fi
        if [ "${l_FLAG}${p_FLAG}" != "yy" ]; then
                echo "WARNING: the -f flag requires -l, and -p;" \
                    "ignoring -f\n" | tee -a $mail_msg_file >> $LOGFILE
                f_FLAG=n
        fi
fi

if [ "$w_FLAG" = "y" -a ! -d $ROOT ]; then
        echo "WARNING: -w specified, but $ROOT does not exist;" \
            "ignoring -w\n" | tee -a $mail_msg_file >> $LOGFILE
        w_FLAG=n
fi

if [ "$t_FLAG" = "n" ]; then
        #
        # We're not doing a tools build, so make sure elfsign(1) is
        # new enough to safely sign non-crypto binaries.  We test
        # debugging output from elfsign to detect the old version.
        #
        newelfsigntest='SUNW_CRYPTO_DEBUG=stderr /usr/bin/elfsign verify \
            -e /usr/lib/security/pkcs11_softtoken.so.1 2>&1 \
            | egrep algorithmOID'
        if [ -z "$newelfsigntest" ]; then
                echo "WARNING: /usr/bin/elfsign out of date;" \
                    "will only sign crypto modules\n" | \
                    tee -a $mail_msg_file >> $LOGFILE
                export ELFSIGN_OBJECT=true
        elif [ "$VERIFY_ELFSIGN" = "y" ]; then
                echo "WARNING: VERIFY_ELFSIGN=y requires" \
                    "the -t flag; ignoring VERIFY_ELFSIGN\n" | \
                    tee -a $mail_msg_file >> $LOGFILE
        fi
fi

[ "$O_FLAG" = y ] && MULTI_PROTO=yes

case $MULTI_PROTO in
yes|no) ;;
*)
        echo "WARNING: MULTI_PROTO is \"$MULTI_PROTO\"; " \
            "should be \"yes\" or \"no\"." | tee -a $mail_msg_file >> $LOGFILE
        echo "Setting MULTI_PROTO to \"no\".\n" | \
            tee -a $mail_msg_file >> $LOGFILE
        export MULTI_PROTO=no
        ;;
esac

echo "\n==== Build version ====\n" | tee -a $mail_msg_file >> $LOGFILE
echo $VERSION | tee -a $mail_msg_file >> $LOGFILE

# Save the current proto area if we're comparing against the last build
if [ "$w_FLAG" = "y" -a -d "$ROOT" ]; then
    if [ -d "$ROOT.prev" ]; then
        rm -rf $ROOT.prev
    fi
    mv $ROOT $ROOT.prev
fi

# Same for non-DEBUG proto area
```

```
1675 if [ "$w_FLAG" = "y" -a "$MULTI_PROTO" = yes -a -d "$ROOT-nd" ]; then
1676         if [ -d "$ROOT-nd.prev" ]; then
1677                 rm -rf $ROOT-nd.prev
1678         fi
1679         mv $ROOT-nd $ROOT-nd.prev
1680 fi

1682 #
1683 # Echo the SCM type of the parent workspace, this can't just be which_scm
1684 # as that does not know how to identify various network repositories.
1685 #
1686 function parent_wstype {
1687         typeset scm_type junk

1689         CODEMGR_WS="$BRINGOVER_WS" "$WHICH_SCM" 2>/dev/null \
1690             | read scm_type junk
1691         if [[ -z "$scm_type" || "$scm_type" == unknown ]]; then
1692                 # Probe BRINGOVER_WS to determine its type
1693                 if [[ $BRINGOVER_WS == svn*://* ]]; then
1694                         scm_type="subversion"
1695                 elif [[ $BRINGOVER_WS == file://* ]] &&
1696                     egrep -s "This is a Subversion repository" \
1697                     ${BRINGOVER_WS#file://}/README.txt 2> /dev/null; then
1698                         scm_type="subversion"
1699                 elif [[ $BRINGOVER_WS == ssh://* ]]; then
1700                         scm_type="mercurial"
1701                 elif [[ $BRINGOVER_WS == http://* ]] && \
1702                     wget -q -O- --save-headers "$BRINGOVER_WS/?cmd=heads" | \
1703                     egrep -s "application/mercurial" 2> /dev/null; then
1704                         scm_type="mercurial"
1705                 elif svn info $BRINGOVER_WS > /dev/null 2>&1; then
1706                         scm_type="subversion"
1707                 else
1708                         scm_type="none"
1709                 fi
1710         fi

1712         # fold both unsupported and unrecognized results into "none"
1713         case "$scm_type" in
1714         none|subversion|teamware|mercurial)
1715                 ;;
1716         *)      scm_type=none
1717                 ;;
1718         esac

1720         echo $scm_type
1721 }
______unchanged_portion_omitted_

1742 SCM_TYPE=$(child_wstype)

1744 #
1745 #       Decide whether to clobber
1746 #
1747 if [ "$i_FLAG" = "n" -a -d "$SRC" ]; then
1748         echo "\n==== Make clobber at 'date' ====\n" >> $LOGFILE

1750         cd $SRC
1751         # remove old clobber file
1752         rm -f $SRC/clobber.out
1753         rm -f $SRC/clobber-${MACH}.out

1755         # Remove all .make.state* files, just in case we are restarting
1756         # the build after having interrupted a previous 'make clobber'.
1757         find . \( -name SCCS -o -name .hg -o -name .svn -o -name .git \
1758                 -o -name 'interfaces.*' \) -prune \
```



```
1759                 -o -name '.make.*' -print | xargs rm -f

1761         $MAKE -ek clobber 2>&1 | tee -a $SRC/clobber-${MACH}.out >> $LOGFILE
1762         echo "\n==== Make clobber ERRORS ====\n" >> $mail_msg_file
1763         grep "$MAKE:" $SRC/clobber-${MACH}.out |
1764                 egrep -v "Ignoring unknown host" \
1765                 >> $mail_msg_file

1767         if [[ "$t_FLAG" = "y" ]]; then
1995         if [[ "$t_FLAG" = "y" || "$O_FLAG" = "y" ]]; then
1768                 echo "\n==== Make tools clobber at 'date' ====\n" >> $LOGFILE
1769                 cd ${TOOLS}
1770                 rm -f ${TOOLS}/clobber-${MACH}.out
1771                 $MAKE TOOLS_PROTO=$TOOLS_PROTO -ek clobber 2>&1 | \
1772                         tee -a ${TOOLS}/clobber-${MACH}.out >> $LOGFILE
1773                 echo "\n==== Make tools clobber ERRORS ====\n" \
1774                         >> $mail_msg_file
1775                 grep "$MAKE:" ${TOOLS}/clobber-${MACH}.out \
1776                         >> $mail_msg_file
1777                 rm -rf ${TOOLS_PROTO}
1778                 mkdir -p ${TOOLS_PROTO}
1779         fi

1781         typeset roots=$(allprotos)
1782         echo "\n\nClearing $roots" >> "$LOGFILE"
1783         rm -rf $roots

1785         # Get back to a clean workspace as much as possible to catch
1786         # problems that only occur on fresh workspaces.
1787         # Remove all .make.state* files, libraries, and .o's that may
1788         # have been omitted from clobber.  A couple of libraries are
1789         # under source code control, so leave them alone.
1790         # We should probably blow away temporary directories too.
1791         cd $SRC
1792         find $relsrcdirs \( -name SCCS -o -name .hg -o -name .svn \
1793             -o -name .git -o -name 'interfaces.*' \) -prune -o \
1794             \( -name '.make.*' -o -name 'lib*.a' -o -name 'lib*.so*' -o \
1795               -name '*.o' \) -print | \
1796             grep -v 'tools/ctf/dwarf/.*/libdwarf' | xargs rm -f
1797 else
1798         echo "\n==== No clobber at 'date' ====\n" >> $LOGFILE
1799 fi

1801 type bringover_teamware > /dev/null 2>&1 || function bringover_teamware {
1802         # sleep on the parent workspace's lock
1803         while egrep -s write $BRINGOVER_WS/Codemgr_wsdata/locks
1804         do
1805                 sleep 120
1806         done

1808         if [[ -z $BRINGOVER ]]; then
1809                 BRINGOVER=$TEAMWARE/bin/bringover
1810         fi

1812         staffer $BRINGOVER -c "nightly update" -p $BRINGOVER_WS \
1813             -w $CODEMGR_WS $BRINGOVER_FILES < /dev/null 2>&1 ||
1814                 touch $TMPDIR/bringover_failed

1816         staffer bringovercheck $CODEMGR_WS >$TMPDIR/bringovercheck.out 2>&1
1817         if [ -s $TMPDIR/bringovercheck.out ]; then
1818                 echo "\n==== POST-BRINGOVER CLEANUP NOISE ====\n"
1819                 cat $TMPDIR/bringovercheck.out
1820         fi
1821 }
______unchanged_portion_omitted_
```

```
1991 #
1992 #       Decide whether to bringover to the codemgr workspace
1993 #
1994 if [ "$n_FLAG" = "n" ]; then
1995         PARENT_SCM_TYPE=$(parent_wstype)

1997         if [[ $SCM_TYPE != none && $SCM_TYPE != $PARENT_SCM_TYPE ]]; then
1998                 echo "cannot bringover from $PARENT_SCM_TYPE to $SCM_TYPE, " \
1999                     "quitting at ‘date‘." | tee -a $mail_msg_file >> $LOGFILE
2000                 exit 1
2001         fi

2003         run_hook PRE_BRINGOVER

2005         echo "\n==== bringover to $CODEMGR_WS at ‘date‘ ====\n" >> $LOGFILE
2006         echo "\n==== BRINGOVER LOG ====\n" >> $mail_msg_file

2008         eval "bringover_${PARENT_SCM_TYPE}" 2>&1 |
2009                 tee -a $mail_msg_file >> $LOGFILE

2011         if [ -f $TMPDIR/bringover_failed ]; then
2012                 rm -f $TMPDIR/bringover_failed
2013                 build_ok=n
2014                 echo "trouble with bringover, quitting at ‘date‘." |
2015                         tee -a $mail_msg_file >> $LOGFILE
2016                 exit 1
2017         fi

2019         #
2020         # It’s possible that we used the bringover above to create
2021         # $CODEMGR_WS.  If so, then SCM_TYPE was previously "none,"
2022         # but should now be the same as $BRINGOVER_WS.
2023         #
2024         [[ $SCM_TYPE = none ]] && SCM_TYPE=$PARENT_SCM_TYPE

2026         run_hook POST_BRINGOVER

2028         check_closed_tree

2030 else
2031         echo "\n==== No bringover to $CODEMGR_WS ====\n" >> $LOGFILE
2032 fi

2262 if [[ "$O_FLAG" = y ]]; then
2263         build_ok=n
2264         echo "OpenSolaris binary deliverables need usr/closed." \
2265             | tee -a "$mail_msg_file" >> $LOGFILE
2266         exit 1
2267 fi

2034 # Safeguards
2035 [[ -v CODEMGR_WS ]] || fatal_error "Error: Variable CODEMGR_WS not set."
2036 [[ -d "${CODEMGR_WS}" ]] || fatal_error "Error: ${CODEMGR_WS} is not a directory
2037 [[ -f "${CODEMGR_WS}/usr/src/Makefile" ]] || fatal_error "Error: ${CODEMGR_WS}/u

2039 echo "\n==== Build environment ====\n" | tee -a $build_environ_file >> $LOGFILE

2041 # System
2042 whence uname | tee -a $build_environ_file >> $LOGFILE
2043 uname -a 2>&1 | tee -a $build_environ_file >> $LOGFILE
2044 echo | tee -a $build_environ_file >> $LOGFILE

2046 # make
2047 whence $MAKE | tee -a $build_environ_file >> $LOGFILE
2048 $MAKE -v | tee -a $build_environ_file >> $LOGFILE
2049 echo "number of concurrent jobs = $DMAKE_MAX_JOBS" |
```



```
2050     tee -a $build_environ_file >> $LOGFILE

2052 #
2053 # Report the compiler versions.
2054 #

2056 if [[ ! -f $SRC/Makefile ]]; then
2057         build_ok=n
2058         echo "\nUnable to find \"Makefile\" in $SRC." | \
2059             tee -a $build_environ_file >> $LOGFILE
2060         exit 1
2061 fi

2063 ( cd $SRC
2064   for target in cc-version cc64-version java-version; do
2065         echo
2066         #
2067         # Put statefile somewhere we know we can write to rather than trip
2068         # over a read-only $srcroot.
2069         #
2070         rm -f $TMPDIR/make-state
2071         export SRC
2072         if $MAKE -K $TMPDIR/make-state -e $target 2>/dev/null; then
2073                 continue
2074         fi
2075         touch $TMPDIR/nocompiler
2076   done
2077   echo
2078 ) | tee -a $build_environ_file >> $LOGFILE

2080 if [ -f $TMPDIR/nocompiler ]; then
2081         rm -f $TMPDIR/nocompiler
2082         build_ok=n
2083         echo "Aborting due to missing compiler." |
2084                 tee -a $build_environ_file >> $LOGFILE
2085         exit 1
2086 fi

2088 # as
2089 whence as | tee -a $build_environ_file >> $LOGFILE
2090 as -V 2>&1 | head -1 | tee -a $build_environ_file >> $LOGFILE
2091 echo | tee -a $build_environ_file >> $LOGFILE

2093 # Check that we’re running a capable link-editor
2094 whence ld | tee -a $build_environ_file >> $LOGFILE
2095 LDVER=‘ld -V 2>&1‘
2096 echo $LDVER | tee -a $build_environ_file >> $LOGFILE
2097 LDVER=‘echo $LDVER | sed -e "s/.*-1\.\([0-9]*\).*/\1/"‘
2098 if [ ‘expr $LDVER \< 422‘ -eq 1 ]; then
2099         echo "The link-editor needs to be at version 422 or higher to build" | \
2100             tee -a $build_environ_file >> $LOGFILE
2101         echo "the latest stuff.  Hope your build works." | \
2102             tee -a $build_environ_file >> $LOGFILE
2103 fi

2105 #
2106 # Build and use the workspace’s tools if requested
2107 #
2108 if [[ "$t_FLAG" = "y" ]]; then
2343 if [[ "$t_FLAG" = "y" || "$O_FLAG" = y ]]; then
2109         set_non_debug_build_flags

2111         build_tools ${TOOLS_PROTO}
2112         if [[ $? != 0  && "$t_FLAG" = y ]]; then
2113                 use_tools $TOOLS_PROTO
2114         fi
```

```
fi

#
# copy ihv proto area in addition to the build itself
#
if [ "$X_FLAG" = "y" ]; then
        copy_ihv_proto
fi

# timestamp the start of the normal build; the findunref tool uses it.
touch $SRC/.build.tstamp
if [ "$i_FLAG" = "y" -a "$SH_FLAG" = "y" ]; then
        echo "\n==== NOT Building base OS-Net source ====\n" | \
            tee -a $LOGFILE >> $mail_msg_file
else
        # timestamp the start of the normal build; the findunref tool uses it.
        touch $SRC/.build.tstamp

        normal_build
fi

#
# Generate the THIRDPARTYLICENSE files if needed.  This is done after
# the build, so that dynamically-created license files are there.
# It's done before findunref to help identify license files that need
# to be added to tools/opensolaris/license-list.
#
if [ "$O_FLAG" = y -a "$build_ok" = y ]; then
        echo "\n==== Generating THIRDPARTYLICENSE files ====\n" |
            tee -a "$mail_msg_file" >> "$LOGFILE"

normal_build
        if [ -d $ROOT/licenses/usr ]; then
                ( cd $ROOT/licenses ; \
                    mktpl $SRC/pkg/license-list ) >> "$LOGFILE" 2>&1
                if (( $? != 0 )) ; then
                        echo "Couldn't create THIRDPARTYLICENSE files" |
                            tee -a "$mail_msg_file" >> "$LOGFILE"
                fi
        else
                echo "No licenses found under $ROOT/licenses" |
                    tee -a "$mail_msg_file" >> "$LOGFILE"
        fi
fi

ORIG_SRC=$SRC
BINARCHIVE=${CODEMGR_WS}/bin-${MACH}.cpio.Z

if [ "$SE_FLAG" = "y" -o "$SD_FLAG" = "y" -o "$SH_FLAG" = "y" ]; then
        save_binaries
fi


# EXPORT_SRC comes after CRYPT_SRC since a domestic build will need
# $SRC pointing to the export_source usr/src.

if [ "$SE_FLAG" = "y" -o "$SD_FLAG" = "y" -o "$SH_FLAG" = "y" ]; then
        if [ "$SD_FLAG" = "y" -a $build_ok = y ]; then
            set_up_source_build ${CODEMGR_WS} ${CRYPT_SRC} CRYPT_SRC
        fi

        if [ $build_ok = y ]; then
            set_up_source_build ${CODEMGR_WS} ${EXPORT_SRC} EXPORT_SRC
        fi
fi
```



```
if [ "$SD_FLAG" = "y" -a $build_ok = y ]; then
        # drop the crypt files in place.
        cd ${EXPORT_SRC}
        echo "\nextracting crypt_files.cpio.Z onto export_source.\n" \
            >> ${LOGFILE}
        zcat ${CODEMGR_WS}/crypt_files.cpio.Z | \
            cpio -idmucvB 2>/dev/null >> ${LOGFILE}
        if [ "$?" = "0" ]; then
                echo "\n==== DOMESTIC extraction succeeded ====\n" \
                    >> $mail_msg_file
        else
                echo "\n==== DOMESTIC extraction failed ====\n" \
                    >> $mail_msg_file
        fi

fi

if [ "$SO_FLAG" = "y" -a "$build_ok" = y ]; then
        #
        # Copy the open sources into their own tree.
        # If copy_source fails, it will have already generated an
        # error message and set build_ok=n, so we don't need to worry
        # about that here.
        #
        copy_source $CODEMGR_WS $OPEN_SRCDIR OPEN_SOURCE usr/src
fi

if [ "$SO_FLAG" = "y" -a "$build_ok" = y ]; then
        SRC=$OPEN_SRCDIR/usr/src
fi

if is_source_build && [ $build_ok = y ] ; then
        # remove proto area(s) here, since we don't clobber
        rm -rf 'allprotos'
        if [ "$t_FLAG" = "y" ]; then
                set_non_debug_build_flags
                ORIG_TOOLS=$TOOLS
                #
                # SRC was set earlier to point to the source build
                # source tree (e.g., $EXPORT_SRC).
                #
                TOOLS=${SRC}/tools
                TOOLS_PROTO=${TOOLS}/${TOOLS_PROTO_REL}; export TOOLS_PROTO
                build_tools ${TOOLS_PROTO}
                if [[ $? != 0 ]]; then
                        use_tools ${TOOLS_PROTO}
                fi
        fi

        normal_build
fi

#
# There are several checks that need to look at the proto area, but
# they only need to look at one, and they don't care whether it's
# DEBUG or non-DEBUG.
#
if [[ "$MULTI_PROTO" = yes && "$D_FLAG" = n ]]; then
        checkroot=$ROOT-nd
else
        checkroot=$ROOT
fi

if [ "$build_ok" = "y" ]; then
        echo "\n==== Creating protolist system file at 'date' ====" \
                >> $LOGFILE
```

```
2147         protolist $checkroot > $ATLOG/proto_list_${MACH}
2148         echo "==== protolist system file created at `date` ====\n" \
2149                 >> $LOGFILE

2151         if [ "$N_FLAG" != "y" ]; then

2153                 E1=
2154                 f1=
2155                 if [ -d "$SRC/pkgdefs" ]; then
2156                         f1="$SRC/pkgdefs/etc/exception_list_$MACH"
2157                         if [ "$X_FLAG" = "y" ]; then
2158                                 f1="$f1 $IA32_IHV_WS/usr/src/pkgdefs/etc/excepti
2159                         fi
2160                 fi

2162                 for f in $f1; do
2163                         if [ -f "$f" ]; then
2164                                 E1="$E1 -e $f"
2165                         fi
2166                 done

2168                 E2=
2169                 f2=
2170                 if [ -d "$SRC/pkg" ]; then
2171                         f2="$f2 exceptions/packaging"
2172                 fi

2174                 for f in $f2; do
2175                         if [ -f "$f" ]; then
2176                                 E2="$E2 -e $f"
2177                         fi
2178                 done

2180                 if [ -f "$REF_PROTO_LIST" ]; then
2181                         #
2182                         # For builds that copy the IHV proto area (-X), add the
2183                         # IHV proto list to the reference list if the reference
2184                         # was built without -X.
2185                         #
2186                         # For builds that don't copy the IHV proto area, add the
2187                         # IHV proto list to the build's proto list if the
2188                         # reference was built with -X.
2189                         #
2190                         # Use the presence of the first file entry of the cached
2191                         # IHV proto list in the reference list to determine
2192                         # whether it was built with -X or not.
2193                         #
2194                         IHV_REF_PROTO_LIST=$SRC/pkg/proto_list_ihv_$MACH
2195                         grepfor=$(nawk '$1 == "f" { print $2; exit }' \
2196                                 $IHV_REF_PROTO_LIST 2> /dev/null)
2197                         if [ $? = 0 -a -n "$grepfor" ]; then
2198                                 if [ "$X_FLAG" = "y" ]; then
2199                                         grep -w "$grepfor" \
2200                                                 $REF_PROTO_LIST > /dev/null
2201                                         if [ ! "$?" = "0" ]; then
2202                                                 REF_IHV_PROTO="-d $IHV_REF_PROTO
2203                                         fi
2204                                 else
2205                                         grep -w "$grepfor" \
2206                                                 $REF_PROTO_LIST > /dev/null
2207                                         if [ "$?" = "0" ]; then
2208                                                 IHV_PROTO_LIST="$IHV_REF_PROTO_L
2209                                         fi
2210                                 fi
2211                         fi
2212                 fi
```



```
2213         fi

2215         if [ "$N_FLAG" != "y" -a -f $SRC/pkgdefs/Makefile ]; then
2216                 echo "\n==== Impact on SVr4 packages ====\n" >> $mail_msg_file
2217                 #
2218                 # Compare the build's proto list with current package
2219                 # definitions to audit the quality of package
2220                 # definitions and makefile install targets. Use the
2221                 # current exception list.
2222                 #
2223                 PKGDEFS_LIST=""
2224                 for d in $abssrcdirs; do
2225                         if [ -d $d/pkgdefs ]; then
2226                                 PKGDEFS_LIST="$PKGDEFS_LIST -d $d/pkgdefs"
2227                         fi
2228                 done
2229                 if [ "$X_FLAG" = "y" -a \
2230                     -d $IA32_IHV_WS/usr/src/pkgdefs ]; then
2231                         PKGDEFS_LIST="$PKGDEFS_LIST -d $IA32_IHV_WS/usr/src/pkgd
2232                 fi
2233                 $PROTOCMPTERSE \
2234                     "Files missing from the proto area:" \
2235                     "Files missing from packages:" \
2236                     "Inconsistencies between pkgdefs and proto area:" \
2237                     ${E1} \
2238                     ${PKGDEFS_LIST} \
2239                     $ATLOG/proto_list_${MACH} \
2240                     >> $mail_msg_file
2241         fi

2243         if [ "$N_FLAG" != "y" -a -d $SRC/pkg ]; then
2244                 echo "\n==== Validating manifests against proto area ====\n" \
2245                     >> $mail_msg_file
2246                 ( cd $SRC/pkg ; $MAKE -e protocmp ROOT="$checkroot" ) \
2247                     >> $mail_msg_file

2249         fi

2251         if [ "$N_FLAG" != "y" -a -f "$REF_PROTO_LIST" ]; then
2252                 echo "\n==== Impact on proto area ====\n" >> $mail_msg_file
2253                 if [ -n "$E2" ]; then
2254                         ELIST=$E2
2255                 else
2256                         ELIST=$E1
2257                 fi
2258                 $PROTOCMPTERSE \
2259                         "Files in yesterday's proto area, but not today's:" \
2260                         "Files in today's proto area, but not yesterday's:" \
2261                         "Files that changed between yesterday and today:" \
2262                         ${ELIST} \
2263                         -d $REF_PROTO_LIST \
2264                         $REF_IHV_PROTO \
2265                         $ATLOG/proto_list_${MACH} \
2266                         $IHV_PROTO_LIST \
2267                         >> $mail_msg_file
2268         fi
2269 fi

2271 if [ "$u_FLAG" = "y"  -a "$build_ok" = "y" ]; then
2272         staffer cp $ATLOG/proto_list_${MACH} \
2273                 $PARENT_WS/usr/src/proto_list_${MACH}
2274 fi

2276 # Update parent proto area if necessary. This is done now
2277 # so that the proto area has either DEBUG or non-DEBUG kernels.
2278 # Note that this clears out the lock file, so we can dispense with
```

```
# the variable now.
if [ "$U_FLAG" = "y" -a "$build_ok" = "y" ]; then
        echo "\n==== Copying proto area to $NIGHTLY_PARENT_ROOT ====\n" | \
            tee -a $LOGFILE >> $mail_msg_file
        rm -rf $NIGHTLY_PARENT_ROOT/*
        unset Ulockfile
        mkdir -p $NIGHTLY_PARENT_ROOT
        if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
                ( cd $ROOT; tar cf - . |
                    ( cd $NIGHTLY_PARENT_ROOT;  umask 0; tar xpf - ) ) 2>&1 |
                    tee -a $mail_msg_file >> $LOGFILE
        fi
        if [[ "$MULTI_PROTO" = yes && "$F_FLAG" = n ]]; then
                rm -rf $NIGHTLY_PARENT_ROOT-nd/*
                mkdir -p $NIGHTLY_PARENT_ROOT-nd
                cd $ROOT-nd
                ( tar cf - . |
                    ( cd $NIGHTLY_PARENT_ROOT-nd; umask 0; tar xpf - ) ) 2>&1 |
                    tee -a $mail_msg_file >> $LOGFILE
        fi
        if [ -n "${NIGHTLY_PARENT_TOOLS_ROOT}" ]; then
                echo "\n==== Copying tools proto area to $NIGHTLY_PARENT_TOOLS_R
                    tee -a $LOGFILE >> $mail_msg_file
                rm -rf $NIGHTLY_PARENT_TOOLS_ROOT/*
                mkdir -p $NIGHTLY_PARENT_TOOLS_ROOT
                if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
                        ( cd $TOOLS_PROTO; tar cf - . |
                            ( cd $NIGHTLY_PARENT_TOOLS_ROOT;
                            umask 0; tar xpf - ) ) 2>&1 |
                            tee -a $mail_msg_file >> $LOGFILE
                fi
        fi
fi

#
# ELF verification: ABI (-A) and runtime (-r) checks
#
if [[ ($build_ok = y) && ( ($A_FLAG = y) || ($r_FLAG = y) ) ]]; then
        # Directory ELF-data.$MACH holds the files produced by these tests.
        elf_ddir=$SRC/ELF-data.$MACH

        # If there is a previous ELF-data backup directory, remove it. Then,
        # rotate current ELF-data directory into its place and create a new
        # empty directory
        rm -rf $elf_ddir.ref
        if [[ -d $elf_ddir ]]; then
                mv $elf_ddir $elf_ddir.ref
        fi
        mkdir -p $elf_ddir

        # Call find_elf to produce a list of the ELF objects in the proto area.
        # This list is passed to check_rtime and interface_check, preventing
        # them from separately calling find_elf to do the same work twice.
        find_elf -fr $checkroot > $elf_ddir/object_list

        if [[ $A_FLAG = y ]]; then
                echo "\n==== Check versioning and ABI information ====\n"  | \
                    tee -a $LOGFILE >> $mail_msg_file

                # Produce interface description for the proto. Report errors.
                interface_check -o -w $elf_ddir -f object_list \
                        -i interface -E interface.err
                if [[ -s $elf_ddir/interface.err ]]; then
                        tee -a $LOGFILE < $elf_ddir/interface.err \
                                >> $mail_msg_file
                fi
```



```
                # If ELF_DATA_BASELINE_DIR is defined, compare the new interface
                # description file to that from the baseline gate. Issue a
                # warning if the baseline is not present, and keep going.
                if [[ "$ELF_DATA_BASELINE_DIR" != '' ]]; then
                        base_ifile="$ELF_DATA_BASELINE_DIR/interface"

                        echo "\n==== Compare versioning and ABI information" \
                            "to baseline ====\n"  | \
                            tee -a $LOGFILE >> $mail_msg_file
                        echo "Baseline:  $base_ifile\n" >> $LOGFILE

                        if [[ -f $base_ifile ]]; then
                                interface_cmp -d -o $base_ile \
                                    $elf_ddir/interface > $elf_ddir/interface.cm
                                if [[ -s $elf_ddir/interface.cmp ]]; then
                                        echo | tee -a $LOGFILE >> $mail_msg_file
                                        tee -a $LOGFILE < \
                                            $elf_ddir/interface.cmp \
                                            >> $mail_msg_file
                                fi
                        else
                                echo "baseline not available. comparison" \
                                    "skipped" | \
                                    tee -a $LOGFILE >> $mail_msg_file
                        fi

                fi
        fi

        if [[ $r_FLAG = y ]]; then
                echo "\n==== Check ELF runtime attributes ====\n" | \
                    tee -a $LOGFILE >> $mail_msg_file

                # If we're doing a DEBUG build the proto area will be left
                # with debuggable objects, thus don't assert -s.
                if [[ $D_FLAG = y ]]; then
                        rtime_sflag=""
                else
                        rtime_sflag="-s"
                fi
                check_rtime -i -m -v $rtime_sflag -o -w $elf_ddir \
                        -D object_list  -f object_list -E runtime.err \
                        -I runtime.attr.raw

                # check_rtime -I output needs to be sorted in order to
                # compare it to that from previous builds.
                sort $elf_ddir/runtime.attr.raw > $elf_ddir/runtime.attr
                rm $elf_ddir/runtime.attr.raw

                # Report errors
                if [[ -s $elf_ddir/runtime.err ]]; then
                        tee -a $LOGFILE < $elf_ddir/runtime.err \
                                >> $mail_msg_file
                fi

                # If there is an ELF-data directory from a previous build,
                # then diff the attr files. These files contain information
                # about dependencies, versioning, and runpaths. There is some
                # overlap with the ABI checking done above, but this also
                # flushes out non-ABI interface differences along with the
                # other information.
                echo "\n==== Diff ELF runtime attributes" \
                    "(since last build) ====\n" | \
                    tee -a $LOGFILE >> $mail_msg_file >> $mail_msg_file
```

```
2411                 if [[ -f $elf_ddir.ref/runtime.attr ]]; then
2412                         diff $elf_ddir.ref/runtime.attr \
2413                                 $elf_ddir/runtime.attr \
2414                                 >> $mail_msg_file
2415                 fi
2416         fi

2418         # If -u set, copy contents of ELF-data.$MACH to the parent workspace.
2419         if [[ "$u_FLAG" = "y" ]]; then
2420                 p_elf_ddir=$PARENT_WS/usr/src/ELF-data.$MACH

2422                 # If parent lacks the ELF-data.$MACH directory, create it
2423                 if [[ ! -d $p_elf_ddir ]]; then
2424                         staffer mkdir -p $p_elf_ddir
2425                 fi

2427                 # These files are used asynchronously by other builds for ABI
2428                 # verification, as above for the -A option. As such, we require
2429                 # the file replacement to be atomic. Copy the data to a temp
2430                 # file in the same filesystem and then rename into place.
2431                 (
2432                         cd $elf_ddir
2433                         for elf_dfile in *; do
2434                                 staffer cp $elf_dfile \
2435                                     ${p_elf_ddir}/${elf_dfile}.new
2436                                 staffer mv -f ${p_elf_ddir}/${elf_dfile}.new \
2437                                     ${p_elf_ddir}/${elf_dfile}
2438                         done
2439                 )
2440         fi
2441 fi

2443 # DEBUG lint of kernel begins

2445 if [ "$i_CMD_LINE_FLAG" = "n" -a "$l_FLAG" = "y" ]; then
2446         if [ "$LINTDIRS" = "" ]; then
2447                 # LINTDIRS="$SRC/uts y $SRC/stand y $SRC/psm y"
2448                 LINTDIRS="$SRC y"
2449         fi
2450         set $LINTDIRS
2451         while [ $# -gt 0 ]; do
2452                 dolint $1 $2; shift; shift
2453         done
2454 else
2455         echo "\n==== No '$MAKE lint' ====\n" >> $LOGFILE
2456 fi

2458 # "make check" begins

2460 if [ "$i_CMD_LINE_FLAG" = "n" -a "$C_FLAG" = "y" ]; then
2461         # remove old check.out
2462         rm -f $SRC/check.out

2464         rm -f $SRC/check-${MACH}.out
2465         cd $SRC
2466         $MAKE -ek check ROOT="$checkroot" 2>&1 | tee -a $SRC/check-${MACH}.out \
2467             >> $LOGFILE
2468         echo "\n==== cstyle/hdrchk errors ====\n" >> $mail_msg_file

2470         grep ":" $SRC/check-${MACH}.out |
2471                 egrep -v "Ignoring unknown host" | \
2472                 sort | uniq >> $mail_msg_file
2473 else
2474         echo "\n==== No '$MAKE check' ====\n" >> $LOGFILE
2475 fi
```



```
2477 echo "\n==== Find core files ====\n" | \
2478     tee -a $LOGFILE >> $mail_msg_file

2480 find $abssrcdirs -name core -a -type f -exec file {} \; | \
2481         tee -a $LOGFILE >> $mail_msg_file

2483 if [ "$f_FLAG" = "y" -a "$build_ok" = "y" ]; then
2484         echo "\n==== Diff unreferenced files (since last build) ====\n" \
2485             | tee -a $LOGFILE >>$mail_msg_file
2486         rm -f $SRC/unref-${MACH}.ref
2487         if [ -f $SRC/unref-${MACH}.out ]; then
2488                 mv $SRC/unref-${MACH}.out $SRC/unref-${MACH}.ref
2489         fi

2491         findunref -S $SCM_TYPE -t $SRC/.build.tstamp -s usr $CODEMGR_WS \
2492             ${TOOLS}/findunref/exception_list 2>> $mail_msg_file | \
2493             sort > $SRC/unref-${MACH}.out

2495         if [ ! -f $SRC/unref-${MACH}.ref ]; then
2496                 cp $SRC/unref-${MACH}.out $SRC/unref-${MACH}.ref
2497         fi

2499         diff $SRC/unref-${MACH}.ref $SRC/unref-${MACH}.out >>$mail_msg_file
2500 fi

2502 #
2503 # Generate the OpenSolaris deliverables if requested.  Some of these
2504 # steps need to come after findunref and are commented below.
2505 #

2839 # If we are doing an OpenSolaris _source_ build (-S O) then we do
2840 # not have usr/closed available to us to generate closedbins from,
2841 # so skip this part.
2842 if [ "$SO_FLAG" = n -a "$O_FLAG" = y -a "$build_ok" = y ]; then
2843         echo "\n==== Generating OpenSolaris tarballs ====\n" | \
2844             tee -a $mail_msg_file >> $LOGFILE

2846         cd $CODEMGR_WS

2848         #
2849         # This step grovels through the package manifests, so it
2850         # must come after findunref.
2851         #
2852         # We assume no DEBUG vs non-DEBUG package content variation
2853         # here; if that changes, then the "make all" in $SRC/pkg will
2854         # need to be moved into the conditionals and repeated for each
2855         # different build.
2856         #
2857         echo "Generating closed binaries tarball(s)..." >> $LOGFILE
2858         closed_basename=on-closed-bins
2859         if [ "$D_FLAG" = y ]; then
2860                 bindrop "$closed_basename" >>"$LOGFILE" 2>&1
2861                 if (( $? != 0 )) ; then
2862                         echo "Couldn't create DEBUG closed binaries." |
2863                             tee -a $mail_msg_file >> $LOGFILE
2864                         build_ok=n
2865                 fi
2866         fi
2867         if [ "$F_FLAG" = n ]; then
2868                 bindrop -n "$closed_basename-nd" >>"$LOGFILE" 2>&1
2869                 if (( $? != 0 )) ; then
2870                         echo "Couldn't create non-DEBUG closed binaries." |
2871                             tee -a $mail_msg_file >> $LOGFILE
2872                         build_ok=n
2873                 fi
2874         fi
```

```
2876         echo "Generating README.opensolaris..." >> $LOGFILE
2877         cat $SRC/tools/opensolaris/README.opensolaris.tmpl | \
2878             mkreadme_osol $CODEMGR_WS/README.opensolaris >> $LOGFILE 2>&1
2879         if (( $? != 0 )) ; then
2880                 echo "Couldn't create README.opensolaris." |
2881                     tee -a $mail_msg_file >> $LOGFILE
2882                 build_ok=n
2883         fi
2884 fi

2507 # Verify that the usual lists of files, such as exception lists,
2508 # contain only valid references to files.  If the build has failed,
2509 # then don't check the proto area.
2510 CHECK_PATHS=${CHECK_PATHS:-y}
2511 if [ "$CHECK_PATHS" = y -a "$N_FLAG" != y ]; then
2512         echo "\n==== Check lists of files ====\n" | tee -a $LOGFILE \
2513                 >>$mail_msg_file
2514         arg=-b
2515         [ "$build_ok" = y ] && arg=
2516         checkpaths $arg $checkroot 2>&1 | tee -a $LOGFILE >>$mail_msg_file
2517 fi

2519 if [ "$M_FLAG" != "y" -a "$build_ok" = y ]; then
2520         echo "\n==== Impact on file permissions ====\n" \
2521                 >> $mail_msg_file

2523         abspkgdefs=
2524         abspkg=
2525         for d in $abssrcdirs; do
2526                 if [ -d "$d/pkgdefs" ]; then
2527                         abspkgdefs="$abspkgdefs $d"
2528                 fi
2529                 if [ -d "$d/pkg" ]; then
2530                         abspkg="$abspkg $d"
2531                 fi
2532         done

2534         if [ -n "$abspkgdefs" ]; then
2535                 pmodes -qvdP \
2536                     'find $abspkgdefs -name pkginfo.tmpl -print -o \
2537                     -name .del\* -prune | sed -e 's:/pkginfo.tmpl$::' | \
2538                     sort -u' >> $mail_msg_file
2539         fi

2541         if [ -n "$abspkg" ]; then
2542                 for d in "$abspkg"; do
2543                         ( cd $d/pkg ; $MAKE -e pmodes ) >> $mail_msg_file
2544                 done
2545         fi
2546 fi

2548 if [ "$w_FLAG" = "y" -a "$build_ok" = "y" ]; then
2549         if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
2550                 do_wsdiff DEBUG $ROOT.prev $ROOT
2551         fi

2553         if [[ "$MULTI_PROTO" = yes && "$F_FLAG" = n ]]; then
2554                 do_wsdiff non-DEBUG $ROOT-nd.prev $ROOT-nd
2555         fi
2556 fi

2558 END_DATE='date'
2559 echo "==== Nightly $maketype build completed: $END_DATE ====" | \
2560     tee -a $LOGFILE >> $build_time_file
```



```
2562 typeset -i10 hours
2563 typeset -Z2 minutes
2564 typeset -Z2 seconds

2566 elapsed_time=$SECONDS
2567 ((hours = elapsed_time / 3600 ))
2568 ((minutes = elapsed_time / 60  % 60))
2569 ((seconds = elapsed_time % 60))

2571 echo "\n==== Total build time ====" | \
2572     tee -a $LOGFILE >> $build_time_file
2573 echo "\nreal    ${hours}:${minutes}:${seconds}" | \
2574     tee -a $LOGFILE >> $build_time_file

2576 if [ "$u_FLAG" = "y" -a "$f_FLAG" = "y" -a "$build_ok" = "y" ]; then
2577         staffer cp ${SRC}/unref-${MACH}.out $PARENT_WS/usr/src/

2579         #
2580         # Produce a master list of unreferenced files -- ideally, we'd
2581         # generate the master just once after all of the nightlies
2582         # have finished, but there's no simple way to know when that
2583         # will be.  Instead, we assume that we're the last nightly to
2584         # finish and merge all of the unref-${MACH}.out files in
2585         # $PARENT_WS/usr/src/.  If we are in fact the final ${MACH} to
2586         # finish, then this file will be the authoritative master
2587         # list.  Otherwise, another ${MACH}'s nightly will eventually
2588         # overwrite ours with its own master, but in the meantime our
2589         # temporary "master" will be no worse than any older master
2590         # which was already on the parent.
2591         #

2593         set -- $PARENT_WS/usr/src/unref-*.out
2594         cp "$1" ${TMPDIR}/unref.merge
2595         shift

2597         for unreffile; do
2598                 comm -12 ${TMPDIR}/unref.merge "$unreffile" > ${TMPDIR}/unref.$$
2599                 mv ${TMPDIR}/unref.$$ ${TMPDIR}/unref.merge
2600         done

2602         staffer cp ${TMPDIR}/unref.merge $PARENT_WS/usr/src/unrefmaster.out
2603 fi

2605 #
2606 # All done save for the sweeping up.
2607 # (whichever exit we hit here will trigger the "cleanup" trap which
2608 # optionally sends mail on completion).
2609 #
2610 if [ "$build_ok" = "y" ]; then
2611         exit 0
2612 fi
2613 exit 1
```