

new/usr/src/Makefile

1

```
*****
7213 Thu Aug 15 11:59:46 2013
new/usr/src/Makefile
4028 remove CLOSED_IS_PRESENT
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 #
23 # Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright (c) 2012 by Delphix. All rights reserved.
25 #
26 #
27 #
28 # Makefile for system source
29 #
30 # include global definitions
31 include Makefile.master
32 #
33 # the Targetdirs file is the AT&T target.dirs file in a makefile format.
34 # it defines TARGETDIRS and ROOTDIRS.
35 include Targetdirs
36 #
37 COMMON_SUBDIRS= uts lib cmd ucplib ucblib ucblib psm man test
38 sparc_SUBDIRS= stand
39 i386_SUBDIRS= grub
40 #
41 #
42 # sparc needs to build stand before psm
43 #
44 $(SPARC_BLD)psm: stand
45 #
46 SUBDIRS= $(COMMON_SUBDIRS) $($MACH)_SUBDIRS)
47 #
48 HDRSUBDIRS= uts head lib cmd
49 #
50 # UCB headers are bug-for-bug compatible and not checkable against the header
51 # standards.
52 #
53 CHKHDRSUBDIRS= head uts lib
54 #
55 #
56 # Headers that can be built in parallel
57 #
58 PARALLEL_HEADERS = sysheaders userheaders libheaders cmdheaders
59 #
60 #
61 # Directories that can be built in parallel
```

new/usr/src/Makefile

2

```
62 #
63 PARALLEL_DIRS = uts lib man
64 #
65 # The check target also causes smf(5) service manifests to be validated.
66 CHKMFSTSUBDIRS= cmd
67 #
68 MSGSUBDIRS= cmd ucblib lib
69 DOMAINS= \
70 SUNW_OST_ADMIN \
71 SUNW_OST_NETRPC \
72 SUNW_OST_OSCMD \
73 SUNW_OST_OSLIB \
74 SUNW_OST_UCBCMD \
75 SUNW_OST_ZONEINFO
76 #
77 MSGDDIRS= $(DOMAINS:%=$(MSGROOT)/%)
78 MSGDIRS= $(MSGROOT) $(MSGDDIRS) $(MSGROOT)/LC_TIME
79 #
80 all := TARGET= all
81 install := TARGET= install
82 install1 := TARGET= install
83 install2 := TARGET= install
84 install_h := TARGET= install_h
85 clean := TARGET= clean
86 clobber := TARGET= clobber
87 check := TARGET= check
88 #
89 .KEEP_STATE:
90 #
91 #
92 # Note: install does not cause a build in pkg. To build packages,
93 # cd pkg and do a 'make install'
94 #
95 #
96 all: mapfiles closedbins sgs .WAIT $(SUBDIRS) pkg
97 #
98 #
99 # The _msg build is a two-step process. First, the _msg dependency
100 # causes recursive makes in $(MSGSUBDIRS), which stages raw message
101 # files in $(ROOT)/catalog. Second, the action from the install
102 # target rule causes those messages to be post-processed from where
103 # they were staged in $(ROOT)/catalog, and the results placed into the
104 # proto area.
105 #
106 # The stage-licenses target causes the license files needed for
107 # packaging to be pulled from $(SRC) and $(CLOSED) and staged in
108 # $(ROOT)/licenses.
109 #
110 install: install1 install2 _msg stage-licenses
111 @cd msg; pwd; $(MAKE) _msg
112 @rm -rf "$(ROOT)/catalog"
113 #
114 stage-licenses: install2
115 @cd pkg; pwd; $(MAKE) stage-licenses
116 #
117 install1: mapfiles closedbins sgs
118 #
119 install2: install1 $(SUBDIRS)
120 #
121 _msg: _msgdirs rootdirs install2 FRC
122 @for m in $(MSGSUBDIRS); do \
123 cd $$m; pwd; $(MAKE) _msg; cd ..; \
124 done
125 #
126 mapfiles: bldtools
127 @cd common/mapfiles; pwd; $(MAKE) install
```

```

129 clean clobber: $(SUBDIRS) head pkg

131 closedbins: bldtools $(ROOTDIRS) FRC
132 @CLOSED_ROOT=$(SON_CLOSED_BINS/root_$(MACH)${RELEASE_BUILD+nd}"; \
133 if [ "$CLOSED_IS_PRESENT" = no ]; then \
133 if [ ! -d "$CLOSED_ROOT" ]; then \
134 $(ECHO) "Error: ON_CLOSED_BINS must point to closed" \
135 "binaries."; \
136 $(ECHO) "root_$(MACH)${RELEASE_BUILD+nd} is not" \
137 "present in $$ON_CLOSED_BINS."; \
138 exit 1; \
139 fi; \
140 $(ECHO) "Copying closed binaries from $$CLOSED_ROOT"; \
141 (cd $$CLOSED_ROOT; \
142 $(TAR) cxF - $(CODEMGR_WS)/exception_lists/closed-bins .) | \
143 (cd $(ROOT); $(TAR) xBpf -); \
144 ( cd $(ROOT); $(CTFSTRIP) $(cd $$CLOSED_ROOT; $(FIND) \
145 ./kernel ./usr/kernel ./platform/*/kernel -type f -a -perm -u+x | \
146 $(EGREP) -vf $(CODEMGR_WS)/exception_lists/closed-bins) )
147 $(EGREP) -vf $(CODEMGR_WS)/exception_lists/closed-bins); \
148 fi

148 #
149 # Declare what parts can be built in parallel
150 # DUMMY at the end is used in case macro expansion produces an empty string to
151 # prevent everything going in parallel
152 #
153 .PARALLEL: $(PARALLEL_HEADERS) DUMMY
154 .PARALLEL: $(PARALLEL_DIRS) DUMMY

156 $(SUBDIRS) head pkg: FRC
157 @cd $@; pwd; $(MAKE) $(TARGET)

159 # librpcsvc has a dependency on headers installed by
160 # userheaders, hence the .WAIT before libheaders.
161 sgs: rootdirs .WAIT sysheaders userheaders .WAIT \
162 libheaders cmdheaders

164 #
165 # Top-level setup target to setup the development environment that includes
166 # headers, tools and generated mapfiles. For open-only builds (i.e.: source
167 # trees w/o usr/closed), this also depends on the closedbins target (above)
168 # in order to properly seed the proto area. Note, although the tools are
169 # dependent on a number of constant mapfiles, the tools themselves are
170 # required to build the generated mapfiles.
171 #
172 setup: closedbins bldtools sgs mapfiles

174 bldtools:
175 @cd tools; pwd; $(MAKE) install

177 # /var/mail/:saved is a special case because of the colon in the name.
178 #
179 rootdirs: $(ROOTDIRS)
180 $(INS) -d -m 775 $(ROOT)/var/mail/:saved

182 lint: FRC
183 $(MAKE) -f Makefile.lint

185 _msgdirs: $(MSGDIRS)

187 $(ROOTDIRS) $(MSGDIRS):
188 $(INS.dir)

190 userheaders: FRC

```

```

191 @cd head; pwd; $(MAKE) install_h

193 libheaders: bldtools
194 @cd lib; pwd; $(MAKE) install_h

196 sysheaders: FRC
197 @cd uts; pwd; $(MAKE) install_h

199 cmdheaders: FRC
200 @cd cmd/fm; pwd; $(MAKE) install_h
201 @cd cmd/mdb; pwd; $(MAKE) install_h

203 check: $(CHKHDRSUBDIRS) $(CHKMFSTSUBDIRS)

205 #
206 # Cross-reference customization: skip all of the subdirectories that
207 # don't contain actual source code.
208 #
209 XRPRUNE = pkg prototypes
210 XRINCDIRS = uts/common head ucbhead

212 cscope.out tags: FRC
213 $(XREF) -f -x $@

215 FRC:

217 #
218 # Targets for reporting compiler versions; nightly uses these.
219 #

221 cc-version:
222 @if $(MACH)_CC -_versions >/dev/null 2>/dev/null; then \
223 $(ECHO) 32-bit compiler; \
224 $(ECHO) $(MACH)_CC; \
225 $(MACH)_CC -_versions 2>&1 | \
226 $(EGREP) '^ (cw|cc|gcc|primary|shadow)'; \
227 else
228 __COMPILER=$(MACH)_CC -_compiler 2>/dev/null || $(TRUE); \
229 if [ -z "$$__COMPILER" ]; then
230 $(ECHO) No 32-bit compiler found;
231 exit 1;
232 else
233 $(ECHO) 32-bit compiler;
234 $(ECHO) $(MACH)_CC;
235 $(ECHO) $__COMPILER;
236 $(MACH)_CC -V 2>&1 | head -1;
237 fi;
238 fi

240 cc64-version:
241 @if $(MACH64)_CC -_versions >/dev/null 2>/dev/null; then \
242 $(ECHO) 64-bit compiler;
243 $(ECHO) $(MACH64)_CC;
244 $(MACH64)_CC -_versions 2>&1 | \
245 $(EGREP) '^ (cw|cc|gcc|primary|shadow)'; \
246 else
247 __COMPILER=$(MACH64)_CC -_compiler 2>/dev/null || $(TRUE); \
248 if [ -z "$$__COMPILER" ]; then
249 $(ECHO) No 64-bit compiler found;
250 exit 1;
251 else
252 $(ECHO) 64-bit compiler;
253 $(ECHO) $(MACH64)_CC;
254 $(ECHO) $__COMPILER;
255 $(MACH64)_CC -V 2>&1 | head -1;
256 fi;

```

```
257     fi
259 java-version:
260     @if [ -x "$(JAVAC)" ]; then
261         $(ECHO) "$(JAVAC)";
262         $(JAVA_ROOT)/bin/java -fullversion 2>&1 | head -1;
263     else
264         $(ECHO) No Java compiler found;
265         exit 1;
266     fi
```

new/usr/src/tools/env/developer.sh

1

```
*****
8013 Thu Aug 15 11:59:46 2013
new/usr/src/tools/env/developer.sh
4028 remove CLOSED_IS_PRESENT
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 #
23 # Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
24 #
25 #
26 # Configuration variables for the runtime environment of the nightly
27 # build script and other tools for construction and packaging of releases.
28 # This script is sourced by 'nightly' and 'bldenv' to set up the environment
29 # for the build. This example is suitable for building a developers workspace,
30 # which will contain the resulting packages and archives. It is based off
31 # the onnv release. It sets NIGHTLY_OPTIONS to make nightly do:
32 # check ELF ABI/versioning (-A)
33 # runs 'make check' (-C)
34 # DEBUG and non-DEBUG builds (-D)
35 # runs lint in usr/src (-l plus the LINTDIRS variable)
36 # sends mail on completion (-m and the MAILTO variable)
37 # creates packages for PIT/RE (-p)
38 # checks for changes in ELF runpaths (-r)
39 #
40 NIGHTLY_OPTIONS="-ACDlmp"; export NIGHTLY_OPTIONS
41 #
42 # This is a variable for the rest of the script - GATE doesn't matter to
43 # nightly itself
44 GATE=onnv-bugfixes; export GATE
45 #
46 # CODEMGR_WS - where is your workspace at (or what should nightly name it)
47 CODEMGR_WS="/builds/$GATE"; export CODEMGR_WS
48 #
49 # PARENT_WS is used to determine the parent of this workspace. This is
50 # for the options that deal with the parent workspace (such as where the
51 # proto area will go).
52 #
53 # If you use this, it must be local (or nfs): nightly cannot copy
54 # over ssh or http.
55 PARENT_WS="/ws/onnv-gate"; export PARENT_WS
56 #
57 # CLONE_WS is the workspace nightly should do a bringover from.
58 CLONE_WS="ssh://anonhg@onnv.sfbay.sun.com/./export/onnv-clone"; export CLONE_WS
59 #
60 # CLOSED_CLONE_WS is the workspace from which nightly should acquire
61 # the usr/closed tree.
```

new/usr/src/tools/env/developer.sh

2

```
62 CLOSED_CLONE_WS="${CLONE_WS}/usr/closed"; export CLOSED_CLONE_WS
63 #
64 # This flag controls whether to build the closed source. If
65 # undefined, nightly(1) and bldenv(1) will set it according to whether
66 # the closed source tree is present. CLOSED_IS_PRESENT="no" means not
67 # building the closed sources.
68 # CLOSED_IS_PRESENT="yes"; export CLOSED_IS_PRESENT
69 #
70 #
71 # The bringover, if any, is done as STAFFER.
72 # Set STAFFER to your own login as gatekeeper or developer
73 # The point is to use group "staff" and avoid referencing the parent
74 # workspace as root.
75 # Some scripts optionally send mail messages to MAILTO.
76 #
77 STAFFER=nobody; export STAFFER
78 MAILTO=$STAFFER; export MAILTO
79 #
80 # The project (see project(4)) under which to run this build. If not
81 # specified, the build is simply run in a new task in the current project.
82 BUILD_PROJECT=; export BUILD_PROJECT
83 #
84 # You should not need to change the next four lines
85 LOCKNAME="basename $CODEMGR_WS_nightly.lock"; export LOCKNAME
86 ATLOG="$CODEMGR_WS/log"; export ATLOG
87 LOGFILE="$ATLOG/nightly.log"; export LOGFILE
88 MACH='uname -p'; export MACH
89 #
90 # When the -A flag is specified, and ELF_DATA_BASELINE_DIR is defined,
91 # the ELF interface description file resulting from the build is compared
92 # to that from the specified directory. This ensures that our object
93 # versioning evolves in a backward compatible manner.
94 #
95 # You should not need to change this unless you wish to use locally cached
96 # baseline files. If you use this, it must be local (or nfs): nightly cannot
97 # copy over ssh or http.
98 #
99 ELF_DATA_BASELINE_DIR="/ws/onnv-gate/usr/src/ELF-data-baseline.$MACH"; export E
100 #
101 # This is usually just needed if the closed tree is missing, or when
102 # building a project gate with the -O (cap oh) flag.
103 ON_CRYPT_BIN="$PARENT_WS/packages/$MACH/on-crypto.$MACH.tar.bz2"
104 export ON_CRYPT_BIN
105 #
106 # REF_PROTO_LIST - for comparing the list of stuff in your proto area
107 # with. Generally this should be left alone, since you want to see differences
108 # from your parent (the gate).
109 #
110 REF_PROTO_LIST=$PARENT_WS/usr/src/proto_list_${MACH}; export REF_PROTO_LIST
111 #
112 # build environment variables, including version info for mcs, motd,
113 # motd, uname and boot messages. Mostly you shouldn't change this except
114 # when the release slips (nah) or you move an environment file to a new
115 # release
116 #
117 ROOT="$CODEMGR_WS/proto/root_${MACH}"; export ROOT
118 SRC="$CODEMGR_WS/usr/src"; export SRC
119 VERSION="$GATE"; export VERSION
120 #
121 # the RELEASE and RELEASE_DATE variables are set in Makefile.master;
122 # there might be special reasons to override them here, but that
123 # should not be the case in general
124 #
125 #
126 # RELEASE="5.10.1"; export RELEASE
127 # RELEASE_DATE="October 2007"; export RELEASE_DATE
```

```

123 # proto area in parent for optionally depositing a copy of headers and
124 # libraries corresponding to the protolibs target
125 # not applicable given the NIGHTLY_OPTIONS
126 #
127 PARENT_ROOT=$PARENT_WS/proto/root_${MACH}; export PARENT_ROOT
128 PARENT_TOOLS_ROOT=$PARENT_WS/usr/src/tools/proto/root_${MACH}-nd; export PARENT_TO
129
130 #
131 # Package creation variables. You probably shouldn't change these,
132 # either.
133 #
134 # PKGARCHIVE determines where repositories will be created.
135 #
136 # PKGPUBLISHER* control the publisher settings for those repositories.
137 #
138 PKGARCHIVE=${CODEMGR_WS}/packages/${MACH}/nightly"; export PKGARCHIVE
139 # PKGPUBLISHER_REDIST="on-redist"; export PKGPUBLISHER_REDI
140 # PKGPUBLISHER_NONREDIST="on-extra"; export PKGPUBLISHER_NONR
141
142 # we want make to do as much as it can, just in case there's more than
143 # one problem.
144 MAKEFLAGS=k; export MAKEFLAGS
145
146 # Magic variable to prevent the devpro compilers/teamware from sending
147 # mail back to devpro on every use.
148 UT_NO_USAGE_TRACKING="1"; export UT_NO_USAGE_TRACKING
149
150 # Build tools - don't set these unless you know what you're doing. These
151 # variables allows you to get the compilers and onbld files locally or
152 # through cacheofs. Set BUILD_TOOLS to pull everything from one location.
153 # Alternately, you can set ONBLD_TOOLS to where you keep the contents of
154 # SUNWonbld and SPRO_ROOT to where you keep the compilers.
155 #
156 #BUILD_TOOLS=/opt; export BUILD_TOOLS
157 #ONBLD_TOOLS=/opt/onbld; export ONBLD_TOOLS
158 #SPRO_ROOT=/opt/SUNWspro; export SPRO_ROOT
159
160 # This goes along with lint - it is a series of the form "A [y|n]" which
161 # means "go to directory A and run 'make lint'" Then mail me (y) the
162 # difference in the lint output. 'y' should only be used if the area you're
163 # linting is actually lint clean or you'll get lots of mail.
164 # You shouldn't need to change this though.
165 #LINTDIRS="$SRC y"; export LINTDIRS
166
167 #
168 # Reference to IA32 IHV workspace, proto area and packages
169 #
170 #IA32_IHV_WS=/ws/${GATE}-ihv; export IA32_IHV_WS
171 #IA32_IHV_ROOT=$IA32_IHV_WS/proto/root_i386; export IA32_IHV_ROOT
172 #IA32_IHV_PKGS=$IA32_IHV_WS/packages/i386/nightly; export IA32_IHV_PKGS
173
174 #
175 # Reference to binary-only IA32 IHV packages
176 #
177 #IA32_IHV_BINARY_PKGS=/ws/${GATE}-ihv-bin
178 #export IA32_IHV_BINARY_PKGS
179
180 # Set this flag to 'n' to disable the automatic validation of the dmake
181 # version in use. The default is to check it.
182 #CHECK_DMAKE=y
183
184 # Set this flag to 'n' to disable the use of 'checkpaths'. The default,
185 # if the 'N' option is not specified, is to run this test.
186 #CHECK_PATHS=y

```

```

188 # Set this flag to 'y' to enable the use of elfsigncmp to validate the
189 # output of elfsign. Doing so requires that 't' be set in NIGHTLY_OPTIONS.
190 # The default is to not verify them.
191 #VERIFY_ELFSIGN=n
192
193 # BRINGOVER_FILES is the list of files nightly passes to bringover.
194 # If not set the default is "usr", but it can be used for bringing
195 # over deleted_files or other nifty directories.
196 #BRINGOVER_FILES="usr deleted_files"
197
198 # POST_NIGHTLY can be any command to be run at the end of nightly. See
199 # nightly(1) for interactions between environment variables and this command.
200 #POST_NIGHTLY=

```

new/usr/src/tools/env/gatekeeper.sh

1

```
*****
8626 Thu Aug 15 11:59:46 2013
new/usr/src/tools/env/gatekeeper.sh
4028 remove CLOSED_IS_PRESENT
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 #
23 # Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
24 #
25 #
26 # Configuration variables for the runtime environment of the nightly
27 # build script and other tools for construction and packaging of releases.
28 # This script is sourced by 'nightly' and 'bldenv' to set up the environment
29 # for the build. This example is suitable for building a gate,
30 # which will contain the resulting packages and archives (builds of the gate
31 # are done in children and then the resulting archives, packages, and proto
32 # area are put into the parent for everyone to use). It is based off
33 # the onnv release. It sets NIGHTLY_OPTIONS to make nightly do:
34 # DEBUG and non-DEBUG builds (-D)
35 # creates packages for PIT/RE (-p)
36 # checks for new interfaces in libraries (-A)
37 # runs 'make check' (-C)
38 # runs lint in usr/src (-l plus the LINTDIRS variable)
39 # sends mail on completion (-m and the MAILTO variable)
40 # updates the protolist in the parent for children to compare with (-u)
41 # updates the proto area in the parent when done (-U)
42 # checks for changes in ELF runpaths (-r)
43 # checks for changes in unreferenced files (-f)
44 #
45 NIGHTLY_OPTIONS="-ADclmpuUrf"; export NIGHTLY_OPTIONS
46 #
47 # This is a variable for the rest of the script - GATE doesn't matter to
48 # nightly itself
49 GATE=onnv-gate; export GATE
50 #
51 # CODEMGR_WS - where is your workspace at (or what should nightly name it)
52 # there is only one definition here, which assumes all the gate build machines
53 # (sparc and x86) are set up the same. But remember, this is a script, so
54 # you could look at $MACH or 'uname -n' and set these variables differently.
55 CODEMGR_WS="/builds/$GATE"; export CODEMGR_WS
56 #
57 # PARENT_WS is used to determine the parent of this workspace. This is
58 # for the options that deal with the parent workspace (such as where the
59 # proto area will go).
60 #
61 # If you use this, it must be local (or nfs): nightly cannot copy
```

new/usr/src/tools/env/gatekeeper.sh

2

```
62 # over ssh or http.
63 PARENT_WS="/ws/$GATE"; export PARENT_WS
64 #
65 # CLONE_WS is the workspace nightly should do a bringover from.
66 CLONE_WS="ssh://anonhg@onnv.sfbay.sun.com//export/onnv-clone"; export CLONE_WS
67 #
68 # CLOSED_CLONE_WS is the workspace from which nightly will acquire the
69 # usr/closed tree.
70 CLOSED_CLONE_WS="${CLONE_WS}/usr/closed"
71 export CLOSED_CLONE_WS
72 #
73 # This flag controls whether to build the closed source. If
74 # undefined, nightly(1) and bldenv(1) will set it according to whether
75 # the closed source tree is present. CLOSED_IS_PRESENT="no" means not
76 # building the closed sources.
77 # CLOSED_IS_PRESENT="yes"; export CLOSED_IS_PRESENT
78 #
79 # The bringover, if any, is done as STAFFER.
80 # Set STAFFER to your own login as gatekeeper or integration engineer.
81 # The point is to use group "staff" and avoid referencing the parent
82 # workspace as root.
83 # Some scripts optionally send mail messages to MAILTO.
84 #
85 STAFFER=nobody; export STAFFER
86 MAILTO=$STAFFER; export MAILTO
87 #
88 # The project (see project(4)) under which to run this build. If not
89 # specified, the build is simply run in a new task in the current project.
90 BUILD_PROJECT=; export BUILD_PROJECT
91 #
92 # You should not need to change the next four lines
93 LOCKNAME="'basename $CODEMGR_WS'_nightly.lock"; export LOCKNAME
94 ATLOG="$CODEMGR_WS/log"; export ATLOG
95 LOGFILE="$ATLOG/nightly.log"; export LOGFILE
96 MACH='uname -p'; export MACH
97 #
98 # When the -A flag is specified, and ELF_DATA_BASELINE_DIR is defined,
99 # the ELF interface description file resulting from the build is compared
100 # to that from the specified directory. This ensures that our object
101 # versioning evolves in a backward compatible manner.
102 #
103 # You should not need to change this unless you wish to use locally cached
104 # baseline files. If you use this, it must be local (or nfs): nightly cannot
105 # copy over ssh or http.
106 #
107 ELF_DATA_BASELINE_DIR="/ws/onnv-gate/usr/src/ELF-data-baseline.$MACH"; export E
108 #
109 # This is usually just needed if the closed tree is missing, or when
110 # building a project gate with the -O (cap oh) flag.
111 # ON_CRYPTO_BINS="$PARENT_WS/packages/$MACH/on-crypto.$MACH.tar.bz2"
112 # export ON_CRYPTO_BINS
113 #
114 # REF_PROTO_LIST - for comparing the list of stuff in your proto area
115 # with. Generally this should be left alone, since you want to see differences
116 # between today's build and yesterdays.
117 #
118 #
119 REF_PROTO_LIST=$PARENT_WS/usr/src/proto_list_{$MACH}; export REF_PROTO_LIST
120 #
121 # build environment variables, including version info for mcs, motd,
122 # motd, uname and boot messages. Mostly you shouldn't change this except
123 # when the release slips (nah) or when starting a new release.
124 #
125 #
126 #
127 ROOT="$CODEMGR_WS/proto/root_{$MACH}"; export ROOT
128 SRC="$CODEMGR_WS/usr/src"; export SRC
129 VERSION="$GATE"; export VERSION
```

```

123 #
124 # the RELEASE and RELEASE_DATE variables are set in Makefile.master;
125 # there might be special reasons to override them here, but that
126 # should not be the case in general
127 #
128 # RELEASE="5.10.1";           export RELEASE
129 # RELEASE_DATE="October 2007"; export RELEASE_DATE

131 # proto area in parent for optionally depositing a copy of headers and
132 # libraries corresponding to the protolibs target
133 #
134 PARENT_ROOT=$PARENT_WS/proto/root_${MACH}; export PARENT_ROOT
135 PARENT_TOOLS_ROOT=$PARENT_WS/usr/src/tools/proto/root_${MACH}-nd; export PARENT_TO

137 #
138 # Package creation variables. You probably shouldn't change these,
139 # either.
140 #
141 # PKGARCHIVE determines where repositories will be created.
142 #
143 # PKGPUBLISHER* control the publisher settings for those repositories.
144 #
145 PKGARCHIVE="${PARENT_WS}/packages/${MACH}/nightly"; export PKGARCHIVE
146 # PKGPUBLISHER_REDIST="on-nightly"; export PKGPUBLISHER_REDI
147 # PKGPUBLISHER_NONREDIST="on-extra"; export PKGPUBLISHER_NONR

150 # we want make to do as much as it can, just in case there's more than
151 # one problem. This is especially important with the gate, since multiple
152 # unrelated broken things can be integrated.
153 MAKEFLAGS=k; export MAKEFLAGS

155 # Magic variable to prevent the devpro compilers/teamware from sending
156 # mail back to devpro on every use.
157 UT_NO_USAGE_TRACKING="1"; export UT_NO_USAGE_TRACKING

159 # Build tools - don't set these unless you know what you're doing. These
160 # variables allows you to get the compilers and onbld files locally or
161 # through cacheofs. Set BUILD_TOOLS to pull everything from one location.
162 # Alternately, you can set ONBLD_TOOLS to where you keep the contents of
163 # SUNWonbld and SPRO_ROOT to where you keep the compilers.
164 #
165 #BUILD_TOOLS=/opt; export BUILD_TOOLS
166 #ONBLD_TOOLS=/opt/onbld; export ONBLD_TOOLS
167 #SPRO_ROOT=/opt/SUNspro; export SPRO_ROOT

169 # This goes along with lint - it is a series of the form "A [y|n]" which
170 # means "go to directory A and run 'make lint'" Then mail me (y) the
171 # difference in the lint output. 'y' should only be used if the area you're
172 # linting is actually lint clean or you'll get lots of mail.
173 # You shouldn't need to change this though.
174 #LINTDIRS="$SRC y"; export LINTDIRS

176 #
177 # Reference to IA32 IHV workspace, proto area and packages
178 #
179 #IA32_IHV_WS=/ws/${GATE}-ihv; export IA32_IHV_WS
180 #IA32_IHV_ROOT=$IA32_IHV_WS/proto/root_i386; export IA32_IHV_ROOT
181 #IA32_IHV_PKGS=$IA32_IHV_WS/packages/i386/nightly; export IA32_IHV_PKGS

183 #
184 # Reference to binary-only IA32 IHV packages
185 #
186 #IA32_IHV_BINARY_PKGS=/ws/${GATE}-ihv-bin
187 #export IA32_IHV_BINARY_PKGS

```

```

189 # Set this flag to 'n' to disable the automatic validation of the dmake
190 # version in use. The default is to check it.
191 #CHECK_DMAKE=y

193 # Set this flag to 'n' to disable the use of 'checkpaths'. The default,
194 # if the 'N' option is not specified, is to run this test.
195 #CHECK_PATHS=y

197 # Set this flag to 'y' to enable the use of elfsigncmp to validate the
198 # output of elfsign. Doing so requires that 't' be set in NIGHTLY_OPTIONS.
199 # The default is to not verify them.
200 #VERIFY_ELFSIGN=n

202 # BRINGOVER_FILES is the list of files nightly passes to bringover.
203 # If not set the default is "usr", but it can be used for bringing
204 # over deleted_files or other nifty directories.
205 #BRINGOVER_FILES="usr deleted_files"

207 # POST_NIGHTLY can be any command to be run at the end of nightly. See
208 # nightly(1) for interactions between environment variables and this command.
209 #POST_NIGHTLY=

```

new/usr/src/tools/scripts/Install.sh

1

```
*****
25470 Thu Aug 15 11:59:46 2013
new/usr/src/tools/scripts/Install.sh
4028 remove CLOSED_IS_PRESENT
*****
_____unchanged_portion_omitted_____

690 function copy_kmdb {
691     typeset kmdbtgtmdir=$INSTALL_FILES/platform/$KARCH/$GLOMNAME/misc
692     typeset bitdirs=
693     typeset isadir=
694     typeset b64srcdir=
695     typeset b64tgtmdir=
696     typeset b32srcdir=
697     typeset b32tgtmdir=
698     typeset machdir=
699     typeset platdir=

701     if [[ $KMDB = "no" || ! -d $SRC/cmd/mdb ]] ; then
702         # The kmdb copy was suppressed or the workspace doesn't contain
703         # the mdb subtree. Either way, there's nothing to do.
704         STATE=2
705         save_state
706         return
707     fi

709     if [[ $(mach) = "i386" ]] ; then
710         isadir="intel"
711         b64srcdir="amd64"
712         b64tgtmdir="amd64"
713         b32srcdir="ia32"
714         b32tgtmdir="."
715     else
716         isadir="sparc"
717         b64srcdir="v9"
718         b64tgtmdir="sparcv9"
719         b32srcdir="v7"
720         b32tgtmdir="."
721     fi

723     typeset foundkmdb=no
724     typeset kmdbpath=
725     typeset destdir=

727     platdir=$INSTALL_FILES/platform/$KARCH/$GLOMNAME
728     if [[ $GLOM = "yes" ]] ; then
729         machdir=$platdir
730     else
731         machdir=$INSTALL_FILES/kernel
732     fi

734     srctrees=$SRC
735     if [[ -d $SRC/../closed && "$CLOSED_IS_PRESENT" != no ]] ; then
736         srctrees="$srctrees $SRC/../closed"
737     else
738         if [ -z "$ON_CRYPTOBINS" ] ; then
739             echo "Warning: ON_CRYPTOBINS not set; pre-signed" \
740                 "crypto not provided."
741         fi
742     fi

743     if [[ $WANT64 = "yes" ]] ; then
744         # kmdbmod for sparc and x86 are built and installed
745         # in different places
746         if [[ $(mach) = "i386" ]] ; then
747             kmdbpath=$SRC/cmd/mdb/$isadir/$b64srcdir/kmdb/kmdbmod
748             destdir=$machdir/misc/$b64tgtmdir
```

new/usr/src/tools/scripts/Install.sh

2

```
745     else
746         kmdbpath=$SRC/cmd/mdb/$KARCH/$b64srcdir/kmdb/kmdbmod
747         destdir=$platdir/misc/$b64tgtmdir
748     fi

750     if kmdb_copy_kmdbmod $kmdbpath $destdir ; then
751         foundkmdb="yes"

753     for tree in $srctrees; do
754         kmdb_copy_machkmods \
755             $tree/cmd/mdb/$isadir/$b64srcdir \
756             $machdir/kmdb/$b64tgtmdir
757         kmdb_copy_karchkmods $tree/cmd/mdb/$KARCH \
758             $platdir/kmdb/$b64tgtmdir $b64srcdir
759     done
760 fi
761 fi

763     if [[ $WANT32 = "yes" ]] ; then
764         kmdbpath=$SRC/cmd/mdb/$isadir/$b32srcdir/kmdb/kmdbmod
765         destdir=$machdir/misc/$b32tgtmdir

767     if kmdb_copy_kmdbmod $kmdbpath $destdir ; then
768         foundkmdb="yes"

770     for tree in $srctrees; do
771         kmdb_copy_machkmods \
772             $tree/cmd/mdb/$isadir/$b32srcdir \
773             $machdir/kmdb/$b32tgtmdir
774         kmdb_copy_karchkmods $tree/cmd/mdb/$KARCH \
775             $platdir/kmdb/$b32tgtmdir $b32srcdir
776     done
777 fi
778 fi

780     # A kmdb-less workspace isn't fatal, but it is potentially problematic,
781     # as the changes made to uts may have altered something upon which kmdb
782     # depends. We will therefore remind the user that they haven't built it
783     # yet.
784     if [[ $foundkmdb != "yes" ]] ; then
785         echo "WARNING: kmdb isn't built, and won't be included"
786     fi

788     STATE=2
789     save_state
790     return
791 }
_____unchanged_portion_omitted_____
```


new/usr/src/tools/scripts/bldenv.sh

1

```
*****
12491 Thu Aug 15 11:59:47 2013
new/usr/src/tools/scripts/bldenv.sh
4028 remove CLOSED_IS_PRESENT
*****
_unchanged_portion_omitted_
154 [+SEE ALSO?\bnightly\b(1)]
155 '

157 # main
158 builtin basename

160 # boolean flags (true/false)
161 typeset flags=(
162     typeset c=false
163     typeset f=false
164     typeset d=false
165     typeset O=false
166     typeset o=false
167     typeset t=true
168     typeset s=(
169         typeset e=false
170         typeset h=false
171         typeset d=false
172         typeset o=false
173     )
174 )

176 typeset progname="${basename -- "${0}"}"

178 OPTIND=1
179 SUFFIX="-nd"

181 while getopts -a "${progname}" "${USAGE}" OPT ; do
182     case ${OPT} in
183         c) flags.c=true ;;
184         +c) flags.c=false ;;
185         f) flags.f=true ;;
186         +f) flags.f=false ;;
187         d) flags.d=true SUFFIX="" ;;
188         +d) flags.d=false SUFFIX="-nd" ;;
189         t) flags.t=true ;;
190         +t) flags.t=false ;;
191         S) set_s_flag "$OPTARG" ;;
192         \?) usage ;;
193     esac
194 done
195 shift $((OPTIND-1))

197 # test that the path to the environment-setting file was given
198 if (( $# < 1 )) ; then
199     usage
200 fi

202 # force locale to C
203 export \
204     LC_COLLATE=C \
205     LC_CTYPE=C \
206     LC_MESSAGES=C \
207     LC_MONETARY=C \
208     LC_NUMERIC=C \
209     LC_TIME=C

211 # clear environment variables we know to be bad for the build
212 unset \
213     LD_OPTIONS \
```

new/usr/src/tools/scripts/bldenv.sh

2

```
214     LD_LIBRARY_PATH \
215     LD_AUDIT \
216     LD_BIND_NOW \
217     LD_BREATH \
218     LD_CONFIG \
219     LD_DEBUG \
220     LD_FLAGS \
221     LD_LIBRARY_PATH_64 \
222     LD_NOVERSION \
223     LD_ORIGIN \
224     LD_LOADFLTR \
225     LD_NOAUXFLTR \
226     LD_NOCONFIG \
227     LD_NODIRCONFIG \
228     LD_NOOBJALTER \
229     LD_PRELOAD \
230     LD_PROFILE \
231     CONFIG \
232     GROUP \
233     OWNER \
234     REMOTE \
235     ENV \
236     ARCH \
237     CLASSPATH

239 #
240 # Setup environment variables
241 #
242 if [[ -f /etc/nightly.conf ]]; then
243     source /etc/nightly.conf
244 fi

246 if [[ -f "$1" ]]; then
247     if [[ "$1" == */* ]]; then
248         source "$1"
249     else
250         source "./$1"
251     fi
252 else
253     if [[ -f "/opt/onbld/env/$1" ]]; then
254         source "/opt/onbld/env/$1"
255     else
256         printf \
257             'Cannot find env file as either %s or /opt/onbld/env/%s\n' \
258             "$1" "$1"
259         exit 1
260     fi
261 fi
262 shift

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

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

273 # must match the getopts in nightly.sh
274 OPTIND=1
275 NIGHTLY_OPTIONS="-${NIGHTLY_OPTIONS#-}"
276 while getopts '+0AaBCDdFfGIiLmNnOopRrS:tUuWwXxz' FLAG "$NIGHTLY_OPTIONS"
277 do
278     case "$FLAG" in
279         O) flags.O=true ;;
```

new/usr/src/tools/scripts/bldenv.sh

3

```

280         +O)  flags.o=false ;;
281         o)    flags.o=true  ;;
282         +o)  flags.o=false ;;
283         t)    flags.t=true  ;;
284         +t)  flags.t=false ;;
285         S)    set S_flag "$OPTARG" ;;
286         *)    ;;
287     esac
288 done

290 POUND_SIGN="#"
291 # have we set RELEASE_DATE in our env file?
292 if [ -z "$RELEASE_DATE" ]; then
293     RELEASE_DATE=$(LC_ALL=C date +"%B %Y")
294 fi
295 BUILD_DATE=$(LC_ALL=C date +%Y-%b-%d)
296 BASEWSDIR=$(basename -- "${CODEMGR_WS}")
297 DEV_CM="\@(#)SunOS Internal Development: $LOGNAME $BUILD_DATE [${BASEWSDIR}\\""
298 export DEV_CM RELEASE_DATE POUND_SIGN

300 export INTERNAL_RELEASE_BUILD=

302 print 'Build type is \c'
303 if ${flags.d} ; then
304     print 'DEBUG'
305     unset RELEASE_BUILD
306     unset EXTRA_OPTIONS
307     unset EXTRA_CFLAGS
308 else
309     # default is a non-DEBUG build
310     print 'non-DEBUG'
311     export RELEASE_BUILD=
312     unset EXTRA_OPTIONS
313     unset EXTRA_CFLAGS
314 fi

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

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

321 # Append source version
322 if "${flags.s.e}" ; then
323     VERSION+=":EXPORT"
324     SRC="${EXPORT_SRC}/usr/src"
325 fi
326
327 if "${flags.s.d}" ; then
328     VERSION+=":DOMESTIC"
329     SRC="${EXPORT_SRC}/usr/src"
330 fi

332 if "${flags.s.h}" ; then
333     VERSION+=":HYBRID"
334     SRC="${EXPORT_SRC}/usr/src"
335 fi
336
337 if "${flags.s.o}" ; then
338     VERSION+=":OPEN_ONLY"
339     SRC="${OPEN_SRCDIR}/usr/src"
340 fi

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

```

new/usr/src/tools/scripts/bldenv.sh

4

```

346 fi

348 if [[ -z "$CLOSED_IS_PRESENT" ]]; then
349     if [[ -d $SRC../closed ]]; then
350         export CLOSED_IS_PRESENT="yes"
351     else
352         export CLOSED_IS_PRESENT="no"
353     fi
354 fi

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

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

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

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

361     export CTFCVPTBL="${TOOLS_PROTO}/opt/onbld/bin/ctfcvptbl"
362     export CTFFINDMOD="${TOOLS_PROTO}/opt/onbld/bin/ctffindmod"

364     PATH="${TOOLS_PROTO}/opt/onbld/bin/${MACH}:${PATH}"
365     PATH="${TOOLS_PROTO}/opt/onbld/bin:${PATH}"
366     export PATH
367 fi

369 export DMAKE_MODE=${DMAKE_MODE:-parallel}

371 if "${flags.o}" ; then
372     export CH=
373 else
374     unset CH
375 fi
376 DEF_STRIPFLAG="-s"

378 TMPDIR="/tmp"

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

384 export \
385     PATH TMPDIR \
386     POUND_SIGN \
387     DEF_STRIPFLAG \
388     RELEASE_DATE
389 unset \
390     CFLAGS \
391     LD_LIBRARY_PATH

393 # a la ws
394 ENVLDLIBS1=
395 ENVLDLIBS2=
396 ENVLDLIBS3=
397 ENVCPPFLAGS1=
398 ENVCPPFLAGS2=
399 ENVCPPFLAGS3=
400 ENVCPPFLAGS4=
401 PARENT_ROOT=
402 PARENT_TOOLS_ROOT=

```

```

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

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

413 export TONICBUILD="#"

415 if "${flags.O}" ; then
424     if [[ "$CLOSED_IS_PRESENT" != "yes" ]]; then
416         print "OpenSolaris closed binary generation requires "
417         print "closed tree"
418         exit 1
428     fi
429     print "Generating OpenSolaris deliverables"
430     # We only need CLOSEDROOT in the env for convenience. Makefile.master
431     # figures out what it needs when it matters.
432     export CLOSEDROOT="${ROOT}-closed"
433     export TONICBUILD="#"
419 fi

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

425 export \
426     ENVLDLIBS1 \
427     ENVLDLIBS2 \
428     ENVLDLIBS3 \
429     ENVCPPFLAGS1 \
430     ENVCPPFLAGS2 \
431     ENVCPPFLAGS3 \
432     ENVCPPFLAGS4 \
433     MAKEFLAGS \
434     PARENT_ROOT \
435     PARENT_TOOLS_ROOT

437 printf 'RELEASE      is %s\n' "$RELEASE"
438 printf 'VERSION      is %s\n' "$VERSION"
439 printf 'RELEASE_DATE  is %s\n\n' "$RELEASE_DATE"

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

446 elif "${flags.t}" ; then
447     printf \
448         'The tools can be (re)built with the install target in %s.\n\n' \
449         "${TOOLS}"
450 fi

452 #
453 # place ourselves in a new task, respecting BUILD_PROJECT if set.
454 #
455 /usr/bin/newtask -c $$ "${BUILD_PROJECT:+-p$BUILD_PROJECT}"

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

461     if "${flags.f}" ; then
462         print 'WARNING: -f is ignored when $SHELL is not csh'

```

```

463     fi

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

468 elif "${flags.f}" ; then
469     print 'Using csh -f as shell.'
470     exec csh -f ${@:+-c "$@"}

472 else
473     print 'Using csh as shell.'
474     exec csh ${@:+-c "$@"}
475 fi

477 # not reached

```

new/usr/src/tools/scripts/checkpaths.sh

1

```
*****
3889 Thu Aug 15 11:59:47 2013
new/usr/src/tools/scripts/checkpaths.sh
4028 remove CLOSED_IS_PRESENT
*****
1 #!/bin/ksh -p
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 #
23 #
24 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
25 # Use is subject to license terms.
26 #
27 #
28 # Quis custodiet ipsos custodiet?
29 #
30 if [ -z "$SRC" ]; then
31     SRC=$CODEMGR_WS/usr/src
32 fi
33 #
34 if [ -z "$CODEMGR_WS" -o ! -d "$CODEMGR_WS" -o ! -d "$SRC" ]; then
35     echo "$0: must be run from within a workspace."
36     exit 1
37 fi
38 #
39 cd $CODEMGR_WS || exit 1
40 #
41 # Use -b to tell this script to ignore derived (built) objects.
42 if [ "$1" = "-b" ]; then
43     b_flg=y
44 fi
45 #
46 # Not currently used; available for temporary workarounds.
47 args="-k NEVER_CHECK"
48 #
49 # We intentionally don't depend on $MACH here, and thus no $ROOT. If
50 # a proto area exists, then we use it. This allows this script to be
51 # run against gates (which should contain both SPARC and x86 proto
52 # areas), build workspaces (which should contain just one proto area),
53 # and unbuilt workspaces (which contain no proto areas).
54 if [ "$b_flg" = y ]; then
55     rootlist=
56 elif [ $# -gt 0 ]; then
57     rootlist=$*
58 else
59     rootlist="$CODEMGR_WS/proto/root_sparc $CODEMGR_WS/proto/root_i386"
60 fi
```

new/usr/src/tools/scripts/checkpaths.sh

2

```
62 # If the closed source is not present, then exclude IKE from validation.
63 if [ "$CLOSED_IS_PRESENT" = no ]; then
64     excl="-e ^usr/include/ike/"
65 fi
66 #
67 #
68 #
69 #
70 #
71 #
72 for ROOT in $rootlist
73 do
74     case "$ROOT" in
75         *sparc|*sparc-nd)
76             arch=sparc
77             ;;
78         *i386|*i386-nd)
79             arch=i386
80             ;;
81         *)
82             echo "$ROOT has unknown architecture." >&2
83             exit 1
84             ;;
85     esac
86     if [ -d $ROOT ]; then
87         #
88         # This is the old-style packaging exception list, from
89         # the svr4-specific usr/src/pkgdefs
90         #
91         [ -f $SRC/pkgdefs/etc/exception_list_$arch ] && \
92             validate_paths '-s/\s*'${arch}'$/' \
93                 -e ^usr/include/ike/ -b $ROOT \
94                 validate_paths '-s/\s*'${arch}'$/' $excl -b $ROOT \
95                 $args $SRC/pkgdefs/etc/exception_list_$arch
96         #
97         # These are the new-style packaging exception lists,
98         # from the repository-wide exception_lists/ directory.
99         #
100         e="$CODEMGR_WS/exception_lists/packaging"
101         for f in $e; do
102             if [ -f $f ]; then
103                 nawk 'NF == 1 || /[ ]\+\'$arch'$/ { print; }'
104                 < $f | validate_paths -b $ROOT -n $f
105             fi
106         done
107     fi
108 done
109 #
110 # Two entries in the findunref exception_list deal with things created
111 # by nightly. Otherwise, this test could be run on an unmodified (and
112 # unbuilt) workspace. We handle this by flagging the one that is
113 # present only on a built workspace (./*.out) and the one that's
114 # present only after a run of findunref (./*.ref) with ISUSED, and
115 # disabling all checks of them. The assumption is that the entries
116 # marked with ISUSED are always known to be good, thus the Latin quote
117 # at the top of the file.
118 #
119 # The exception_list is generated from whichever input files are appropriate
120 # for this workspace, so checking it obviates the need to check the inputs.
121 #
122 if [ -r $SRC/tools/findunref/exception_list ]; then
123     validate_paths -k ISUSED -r -e '^*' $SRC/tools/findunref/exception_list
124 fi
125 #
126 if [ -f $SRC/tools/opensolaris/license-list ]; then
127     excl=
128     if [ "$CLOSED_IS_PRESENT" = no ]; then
129         excl="-e ^usr/closed"
130     fi
131     sed -e 's/./descrip/' < $SRC/tools/opensolaris/license-list | \
132         validate_paths -n SRC/tools/opensolaris/license-list \
```

```
118         -e ^usr/closed
125         validate_paths -n SRC/tools/opensolaris/license-list $excl
119 fi

121 # Finally, make sure the that (req|inc).flg files are in good shape.
122 # If SCCS files are not expected to be present, though, then don't
123 # check them.
124 if [ ! -d "$CODEMGR_WS/Codemgr_wsdata" ]; then
125     f_flg='-f'
126 fi
134 # If the closed source is not present, then don't validate it.
135 if [ "$CLOSED_IS_PRESENT" = no ]; then
136     excl="-e ^usr/closed/"
137 fi

128 validate_flg $f_flg -e ^usr/closed/
139 validate_flg $f_flg $excl

130 exit 0
```

new/usr/src/tools/scripts/nightly.sh

1

```
*****
80249 Thu Aug 15 11:59:47 2013
new/usr/src/tools/scripts/nightly.sh
4028 remove CLOSED_IS_PRESENT
*****
_____unchanged_portion_omitted_____

296 #
297 # Mercurial-specific copy code for copy_source().
297 # Mercurial-specific copy code for copy_source(). Handles the
298 # combined open and closed trees.
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     typeset open_top closed_top

307     hg locate -I "$srcroot" | cpio -pd "$dest" >>$LOGFILE 2>&1
309     case $srcroot in
310     usr)
311         open_top=usr
312         if [[ "$CLOSED_IS_PRESENT" = yes ]]; then
313             closed_top=usr/closed
314         fi
315         ;;
316     usr/closed*)
317         if [[ "$CLOSED_IS_PRESENT" = no ]]; then
318             printf "can't copy %s: closed tree not present.\n" \
319                 "$srcroot" | tee -a $mail_msg_file >> $LOGFILE
320             return 1
321         fi
322         closed_top="$srcroot"
323         ;;
324     *)
325         open_top="$srcroot"
326         ;;
327     esac

329     if [[ -n "$open_top" ]]; then
330         hg locate -I "$open_top" | cpio -pd "$dest" >>$LOGFILE 2>&1
338     if [[ -n "$closed_top" ]]; then
339         mkdir -p "$dest/usr/closed" || return 1
340         if [[ "$closed_top" = usr/closed ]]; then
341             (cd usr/closed; hg locate |
342             cpio -pd "$dest/usr/closed") >>$LOGFILE 2>&1
343             if (( $? != 0 )); then
344                 printf "cpio failed for %s/usr/closed\n" \
345                     "$dest" | tee -a $mail_msg_file >> $LOGFILE
346                 return 1
347             fi
348         else
349             # copy subtree of usr/closed
350             closed_top=${closed_top#usr/closed/}
351             (cd usr/closed; hg locate -I "$closed_top" |
352             cpio -pd "$dest/usr/closed") >>$LOGFILE 2>&1
```

new/usr/src/tools/scripts/nightly.sh

2

```
353         if (( $? != 0 )); then
354             printf "cpio failed for %s/usr/closed/%s\n" \
355                 "$dest" "$closed_top" |
356                 tee -a $mail_msg_file >> $LOGFILE
357             return 1
358         fi
359     fi
360 fi

314     return 0
315 }
_____unchanged_portion_omitted_____

846 #
847 # Verify that the closed tree is present if it needs to be.
848 #
849 function check_closed_tree {
850     if [[ ! -d "$ON_CLOSED_BINS" ]]; then
851         echo "ON_CLOSED_BINS must point to the closed binaries tree."
852         echo "If the closed sources are not present," \
853             "ON_CLOSED_BINS"
854         echo "must point to the closed binaries tree."
855     fi
856 }
_____unchanged_portion_omitted_____

1041 OPTIND=1
1042 while getopts +inS:tV: FLAG
1043 do
1044     case $FLAG in
1045     i ) i_FLAG=y; i_CMD_LINE_FLAG=y
1046         ;;
1047     n ) n_FLAG=y
1048         ;;
1049     S )
1050         set_S_flag $OPTARG
1051         ;;
1052     +t ) t_FLAG=n
1053         ;;
1054     V ) V_FLAG=y
1055         V_ARG="$OPTARG"
1056         ;;
1057     \? ) echo "$USAGE"
1058         exit 1
1059         ;;
1060     esac
1061 done

1063 # correct argument count after options
1064 shift `expr $OPTIND - 1`

1066 # test that the path to the environment-setting file was given
1067 if [ $# -ne 1 ]; then
1068     echo "$USAGE"
1069     exit 1
1070 fi

1072 # check if user is running nightly as root
1073 # ISUSER is set non-zero if an ordinary user runs nightly, or is zero
1074 # when root invokes nightly.
1075 /usr/bin/id | grep '^uid=0(' >/dev/null 2>&1
1076 ISUSER=${?}; export ISUSER

1078 #
```

new/usr/src/tools/scripts/nightly.sh

```

1079 # force locale to C
1080 LC_COLLATE=C; export LC_COLLATE
1081 LC_CTYPE=C; export LC_CTYPE
1082 LC_MESSAGES=C; export LC_MESSAGES
1083 LC_MONETARY=C; export LC_MONETARY
1084 LC_NUMERIC=C; export LC_NUMERIC
1085 LC_TIME=C; export LC_TIME

1087 # clear environment variables we know to be bad for the build
1088 unset LD_OPTIONS
1089 unset LD_AUDIT LD_AUDIT_32 LD_AUDIT_64
1090 unset LD_BIND_NOW LD_BIND_NOW_32 LD_BIND_NOW_64
1091 unset LD_BREADTH LD_BREADTH_32 LD_BREADTH_64
1092 unset LD_CONFIG LD_CONFIG_32 LD_CONFIG_64
1093 unset LD_DEBUG LD_DEBUG_32 LD_DEBUG_64
1094 unset LD_DEMANGLE LD_DEMANGLE_32 LD_DEMANGLE_64
1095 unset LD_FLAGS LD_FLAGS_32 LD_FLAGS_64
1096 unset LD_LIBRARY_PATH LD_LIBRARY_PATH_32 LD_LIBRARY_PATH_64
1097 unset LD_LOADFLTR LD_LOADFLTR_32 LD_LOADFLTR_64
1098 unset LD_NOAUDIT LD_NOAUDIT_32 LD_NOAUDIT_64
1099 unset LD_NOAUXFLTR LD_NOAUXFLTR_32 LD_NOAUXFLTR_64
1100 unset LD_NOCONFIG LD_NOCONFIG_32 LD_NOCONFIG_64
1101 unset LD_NODIRCONFIG LD_NODIRCONFIG_32 LD_NODIRCONFIG_64
1102 unset LD_NODIRECT LD_NODIRECT_32 LD_NODIRECT_64
1103 unset LD_NOLAZYLOAD LD_NOLAZYLOAD_32 LD_NOLAZYLOAD_64
1104 unset LD_NOOBJALTER LD_NOOBJALTER_32 LD_NOOBJALTER_64
1105 unset LD_NOVERSION LD_NOVERSION_32 LD_NOVERSION_64
1106 unset LD_ORIGIN LD_ORIGIN_32 LD_ORIGIN_64
1107 unset LD_PRELOAD LD_PRELOAD_32 LD_PRELOAD_64
1108 unset LD_PROFILE LD_PROFILE_32 LD_PROFILE_64

1110 unset CONFIG
1111 unset GROUP
1112 unset OWNER
1113 unset REMOTE
1114 unset ENV
1115 unset ARCH
1116 unset CLASSPATH
1117 unset NAME

1119 #
1120 # To get ONBLD_TOOLS from the environment, it must come from the env file.
1121 # If it comes interactively, it is generally TOOLS_PROTO, which will be
1122 # clobbered before the compiler version checks, which will therefore fail.
1123 #
1124 unset ONBLD_TOOLS

1126 #
1127 # Setup environmental variables
1128 #
1129 if [ -f /etc/nightly.conf ]; then
1130 . /etc/nightly.conf
1131 fi

1133 if [ -f $1 ]; then
1134 if [[ $1 = */* ]]; then
1135 . $1
1136 else
1137 . ./$1
1138 fi
1139 else
1140 if [ -f $OPTHOME/onbld/env/$1 ]; then
1141 . $OPTHOME/onbld/env/$1
1142 else
1143 echo "Cannot find env file as either $1 or $OPTHOME/onbld/env/$1
1144 exit 1

```

3

new/usr/src/tools/scripts/nightly.sh

```

1145 fi
1146 fi

1148 # contents of stdenv.sh inserted after next line:
1149 # STDENV_START
1150 # STDENV_END

1152 # Check if we have sufficient data to continue...
1153 [[ -v CODEMGR_WS ]] || fatal_error "Error: Variable CODEMGR_WS not set."
1154 if [[ "${NIGHTLY_OPTIONS}" =~ ~(F)n ]]; then
1155 # Check if the gate data are valid if we don't do a "bringover" below
1156 [[ -d "${CODEMGR_WS}" ]] || \
1157 fatal_error "Error: ${CODEMGR_WS} is not a directory."
1158 [[ -f "${CODEMGR_WS}/usr/src/Makefile" ]] || \
1159 fatal_error "Error: ${CODEMGR_WS}/usr/src/Makefile not found."
1160 fi

1162 #
1163 # place ourselves in a new task, respecting BUILD_PROJECT if set.
1164 #
1165 if [ -z "$BUILD_PROJECT" ]; then
1166 /usr/bin/newtask -c $$
1167 else
1168 /usr/bin/newtask -c $$ -p $BUILD_PROJECT
1169 fi

1171 ps -o taskid= -p $$ | read build_taskid
1172 ps -o project= -p $$ | read build_project

1174 #
1175 # See if NIGHTLY_OPTIONS is set
1176 #
1177 if [ "$NIGHTLY_OPTIONS" = "" ]; then
1178 NIGHTLY_OPTIONS="-aBm"
1179 fi

1181 #
1182 # If BRINGOVER_WS was not specified, let it default to CLONE_WS
1183 #
1184 if [ "$BRINGOVER_WS" = "" ]; then
1185 BRINGOVER_WS=$CLONE_WS
1186 fi

1188 #
1189 # If CLOSED_BRINGOVER_WS was not specified, let it default to CLOSED_CLONE_WS
1190 #
1191 if [ "$CLOSED_BRINGOVER_WS" = "" ]; then
1192 CLOSED_BRINGOVER_WS=$CLOSED_CLONE_WS
1193 fi

1195 #
1196 # If BRINGOVER_FILES was not specified, default to usr
1197 #
1198 if [ "$BRINGOVER_FILES" = "" ]; then
1199 BRINGOVER_FILES="usr"
1200 fi

1252 #
1253 # If the closed sources are not present, the closed binaries must be
1254 # present for the build to succeed. If there's no pointer to the
1255 # closed binaries, flag that now, rather than forcing the user to wait
1256 # a couple hours (or more) to find out.
1257 #
1258 orig_closed_is_present="$CLOSED_IS_PRESENT"
1202 check_closed_tree

```

4

```

1204 #
1205 # Note: changes to the option letters here should also be applied to the
1206 #       bldenv script. 'd' is listed for backward compatibility.
1207 #
1208 NIGHTLY_OPTIONS=-${NIGHTLY_OPTIONS#-}
1209 OPTIND=1
1210 while getopts +ABCDdFfGgIilMmNnOoPpRrS:TtUuWwXxz FLAG $NIGHTLY_OPTIONS
1211 do
1212     case $FLAG in
1213         A ) A_FLAG=y
1214             #
1215             # If ELF_DATA_BASELINE_DIR is not defined, and we are on SWAN
1216             # (based on CLOSED_IS_PRESENT), then refuse to run. The value
1217             # of ELF version checking is greatly enhanced by including
1218             # the baseline gate comparison.
1219             if [ "$CLOSED_IS_PRESENT" = 'yes' -a \
1220                 "$ELF_DATA_BASELINE_DIR" = '' ]; then
1221                 echo "ELF_DATA_BASELINE_DIR must be set if the A" \
1222                     "flag is present in\nNIGHTLY_OPTIONS and closed" \
1223                     "sources are present. Update environment file." \
1224                     exit 1;
1225             fi
1226             ;;
1227         B ) D_FLAG=y
1228             ;; # old version of D
1229         C ) C_FLAG=y
1230             ;;
1231         D ) D_FLAG=y
1232             ;;
1233         F ) F_FLAG=y
1234             ;;
1235         f ) f_FLAG=y
1236             ;;
1237         G ) u_FLAG=y
1238             ;;
1239         I ) m_FLAG=y
1240             p_FLAG=y
1241             u_FLAG=y
1242             ;;
1243         i ) i_FLAG=y
1244             ;;
1245         l ) l_FLAG=y
1246             ;;
1247         M ) M_FLAG=y
1248             ;;
1249         m ) m_FLAG=y
1250             ;;
1251         N ) N_FLAG=y
1252             ;;
1253         n ) n_FLAG=y
1254             ;;
1255         O ) O_FLAG=y
1256             ;;
1257         o ) o_FLAG=y
1258             ;;
1259         P ) P_FLAG=y
1260             ;; # obsolete
1261         p ) p_FLAG=y
1262             ;;
1263         R ) m_FLAG=y
1264             p_FLAG=y
1265             ;;
1266         r ) r_FLAG=y
1267             ;;
1268         S )
1269             set_S_flag $OPTARG
1270     esac

```

```

1258     ;;
1259     T ) T_FLAG=y
1260     ;; # obsolete
1261     +t ) t_FLAG=n
1262     ;;
1263     U ) if [ -z "${PARENT_ROOT}" ]; then
1264           echo "PARENT_ROOT must be set if the U flag is" \
1265               "present in NIGHTLY_OPTIONS."
1266           exit 1
1267       fi
1268       NIGHTLY_PARENT_ROOT=$PARENT_ROOT
1269       if [ -n "${PARENT_TOOLS_ROOT}" ]; then
1270           NIGHTLY_PARENT_TOOLS_ROOT=$PARENT_TOOLS_ROOT
1271       fi
1272       U_FLAG=y
1273       ;;
1274     u ) u_FLAG=y
1275     ;;
1276     W ) W_FLAG=y
1277     ;;
1278     w ) w_FLAG=y
1279     ;;
1280     X ) # now that we no longer need realmode builds, just
1281         # copy IHV packages. only meaningful on x86.
1282         if [ "$MACH" = "i386" ]; then
1283             X_FLAG=y
1284         fi
1285     ;;
1286     x ) XMOD_OPT="-x"
1287     ;;
1288     \? ) echo "$USAGE"
1289         exit 1
1290     ;;
1291     esac
1292 done
1293
1294 if [ $ISUSER -ne 0 ]; then
1295     if [ "$o_FLAG" = "y" ]; then
1296         echo "Old-style build requires root permission."
1297         exit 1
1298     fi
1299
1300     # Set default value for STAFFER, if needed.
1301     if [ -z "$STAFFER" -o "$STAFFER" = "nobody" ]; then
1302         STAFFER="/usr/xpg4/bin/id -un"
1303     fi
1304     export STAFFER
1305 fi
1306
1307 if [ -z "$MAILTO" -o "$MAILTO" = "nobody" ]; then
1308     MAILTO=$STAFFER
1309     export MAILTO
1310 fi
1311
1312 PATH="$OPTHOME/onbld/bin:$OPTHOME/onbld/bin/${MACH}:/usr/ccs/bin"
1313 PATH="$PATH:$OPTHOME/SUNWsprow/bin:$TEAMWARE/bin:/usr/bin:/usr/sbin:/usr/ucb"
1314 PATH="$PATH:/usr/openwin/bin:/usr/sfw/bin:/opt/sfw/bin:."
1315 export PATH
1316
1317 # roots of source trees, both relative to $SRC and absolute.
1318 relsrcdirs="."
1319 absrsrcdirs="$SRC"
1320 if [[ -d $CODEMGR_WS/usr/closed && "$CLOSED_IS_PRESENT" != no ]]; then
1321     relsrcdirs="$relsrcdirs ../closed"
1322 fi

```



```

1392 abssrkdirs=""
1393 for d in $relsrkdirs; do
1394     abssrkdirs="$abssrkdirs $SRC/$d"
1395 done

1322 unset CH
1323 if [ "$o_FLAG" = "y" ]; then
1324 # root invoked old-style build -- make sure it works as it always has
1325 # by exporting 'CH'. The current Makefile.master doesn't use this, but
1326 # the old ones still do.
1327     PROTOCMPTERSE="protocmp.terse"
1328     CH=
1329     export CH
1330 else
1331     PROTOCMPTERSE="protocmp.terse -gu"
1332 fi
1333 POUND_SIGN="#"
1334 # have we set RELEASE_DATE in our env file?
1335 if [ -z "$RELEASE_DATE" ]; then
1336     RELEASE_DATE=$(LC_ALL=C date +%B %Y)
1337 fi
1338 BUILD_DATE=$(LC_ALL=C date +%Y-%b-%d)
1339 BASEWSDIR=$(basename $CODEMGR_WS)
1340 DEV_CM="\@(#)SunOS Internal Development: $LOGNAME $BUILD_DATE [$BASEWSDIR]\\"

1342 # we export POUND_SIGN, RELEASE_DATE and DEV_CM to speed up the build process
1343 # by avoiding repeated shell invocations to evaluate Makefile.master definitions
1344 # we export o_FLAG and X_FLAG for use by makebfu, and by usr/src/pkg/Makefile
1345 export o_FLAG X_FLAG POUND_SIGN RELEASE_DATE DEV_CM

1347 maketype="distributed"
1348 MAKE=dmake
1349 # get the dmake version string alone
1350 DMAKE_VERSION=$( $MAKE -v )
1351 DMAKE_VERSION=${DMAKE_VERSION##*: }
1352 # focus in on just the dotted version number alone
1353 DMAKE_MAJOR=$( echo $DMAKE_VERSION | \
1354     sed -e 's/.*\([^\.]*\.[^\ ]*\).*$/\1/' )
1355 # extract the second (or final) integer
1356 DMAKE_MINOR=${DMAKE_MAJOR##*.}
1357 DMAKE_MINOR=${DMAKE_MINOR%.*}
1358 # extract the first integer
1359 DMAKE_MAJOR=${DMAKE_MAJOR%.*}
1360 CHECK_DMAKE=${CHECK_DMAKE:-y}
1361 # x86 was built on the 12th, sparc on the 13th.
1362 if [ "$CHECK_DMAKE" = "y" -a \
1363     "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/12" -a \
1364     "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/13" -a \
1365     "$DMAKE_MAJOR" -lt 7 -o \
1366     "$DMAKE_MAJOR" -eq 7 -a "$DMAKE_MINOR" -lt 4 \ ]; then
1367     if [ -z "$DMAKE_VERSION" ]; then
1368         echo "$MAKE is missing."
1369         exit 1
1370     fi
1371     echo `whence $MAKE` version is:
1372     echo " $DMAKE_VERSION"
1373     cat <<EOF

1375 This version may not be safe for use. Either set TEAMWARE to a better
1376 path or (if you really want to use this version of dmake anyway), add
1377 the following to your environment to disable this check:

1379 CHECK_DMAKE=n
1380 EOF
1381     exit 1
1382 fi

```

```

1383 export PATH
1384 export MAKE

1386 if [ [ "$O_FLAG" = y ] ]; then
1387     export TONICBUILD=""
1388 else
1389     export TONICBUILD="#"
1390 fi

1392 if [ "${SUNWSPRO}" != "" ]; then
1393     PATH="${SUNWSPRO}/bin:$PATH"
1394     export PATH
1395 fi

1397 hostname=$(uname -n)
1398 if [ [ $DMAKE_MAX_JOBS != +([0-9]) || $DMAKE_MAX_JOBS -eq 0 ] ]
1399 then
1400     maxjobs=
1401     if [ [ -f $HOME/.make.machines ] ]
1402     then
1403         # Note: there is a hard tab and space character in the [ ]s
1404         # below.
1405         egrep -i "^[ \t]*$hostname[ \t \.]" \
1406             $HOME/.make.machines | read host jobs
1407         maxjobs=${jobs##* }
1408     fi

1410     if [ [ $maxjobs != +([0-9]) || $maxjobs -eq 0 ] ]
1411     then
1412         # default
1413         maxjobs=4
1414     fi

1416     export DMAKE_MAX_JOBS=$maxjobs
1417 fi

1419 DMAKE_MODE=parallel;
1420 export DMAKE_MODE

1422 if [ -z "${ROOT}" ]; then
1423     echo "ROOT must be set."
1424     exit 1
1425 fi

1427 #
1428 # if -V flag was given, reset VERSION to V_ARG
1429 #
1430 if [ "$V_FLAG" = "y" ]; then
1431     VERSION=$V_ARG
1432 fi

1434 #
1435 # Check for IHV root for copying ihv proto area
1436 #
1437 if [ "$X_FLAG" = "y" ]; then
1438     if [ "$IA32_IHV_ROOT" = "" ]; then
1439         echo "IA32_IHV_ROOT: must be set for copying ihv proto"
1440         args_ok=n
1441     fi
1442     if [ ! -d "$IA32_IHV_ROOT" ]; then
1443         echo "$IA32_IHV_ROOT: not found"
1444         args_ok=n
1445     fi
1446     if [ "$IA32_IHV_WS" = "" ]; then
1447         echo "IA32_IHV_WS: must be set for copying ihv proto"
1448         args_ok=n

```

new/usr/src/tools/scripts/nightly.sh

9

```

1449         fi
1450         if [ ! -d "$IA32_IHV_WS" ]; then
1451             echo "$IA32_IHV_WS: not found"
1452             args_ok=n
1453         fi
1454     fi

1456 # Append source version
1457 if [ "$SSE_FLAG" = "y" ]; then
1458     VERSION="${VERSION}:EXPORT"
1459 fi

1461 if [ "$SSD_FLAG" = "y" ]; then
1462     VERSION="${VERSION}:DOMESTIC"
1463 fi

1465 if [ "$SSH_FLAG" = "y" ]; then
1466     VERSION="${VERSION}:MODIFIED_SOURCE_PRODUCT"
1467 fi

1469 if [ "$SSO_FLAG" = "y" ]; then
1470     VERSION="${VERSION}:OPEN_ONLY"
1471 fi

1473 TMPDIR="/tmp/nightly.tmpdir.$$"
1474 export TMPDIR
1475 rm -rf ${TMPDIR}
1476 mkdir -p $TMPDIR || exit 1
1477 chmod 777 $TMPDIR

1479 #
1480 # Keep elfsign's use of pkcs11_softtoken from looking in the user home
1481 # directory, which doesn't always work. Needed until all build machines
1482 # have the fix for 6271754
1483 #
1484 SOFTTOKEN_DIR=$TMPDIR
1485 export SOFTTOKEN_DIR

1487 #
1488 # Tools should only be built non-DEBUG. Keep track of the tools proto
1489 # area path relative to $TOOLS, because the latter changes in an
1490 # export build.
1491 #
1492 # TOOLS_PROTO is included below for builds other than usr/src/tools
1493 # that look for this location. For usr/src/tools, this will be
1494 # overridden on the $MAKE command line in build_tools().
1495 #
1496 TOOLS=${SRC}/tools
1497 TOOLS_PROTO_REL=proto/root_${MACH}-nd
1498 TOOLS_PROTO=${TOOLS}/${TOOLS_PROTO_REL}; export TOOLS_PROTO

1500 unset CFLAGS LD_LIBRARY_PATH LDFLAGS

1502 # create directories that are automatically removed if the nightly script
1503 # fails to start correctly
1504 function newdir {
1505     dir=$1
1506     toadd=
1507     while [ ! -d $dir ]; do
1508         toadd="$dir $toadd"
1509         dir='dirname $dir'
1510     done
1511     torm=
1512     newlist=
1513     for dir in $toadd; do
1514         if staffer mkdir $dir; then

```

new/usr/src/tools/scripts/nightly.sh

10

```

1515         newlist="$ISUSER $dir $newlist"
1516         torm="$dir $torm"
1517     else
1518         [ -z "$storm" ] || staffer rmdir $storm
1519     return 1
1520     fi
1521 done
1522 newdirlist="$newlist $newdirlist"
1523 return 0
1524 }

```

unchanged_portion_omitted

```

2057 type bringover_mercurial > /dev/null 2>&1 || function bringover_mercurial {
2058     typeset -x PATH=$PATH

2060     # If the repository doesn't exist yet, then we want to populate it.
2061     if [[ ! -d $CODEMGR_WS/.hg ]]; then
2062         staffer hg init $CODEMGR_WS
2063         staffer echo "[paths]" > $CODEMGR_WS/.hg/hgrc
2064         staffer echo "default=$BRINGOVER_WS" >> $CODEMGR_WS/.hg/hgrc
2065         touch $TMPDIR/new_repository
2066     fi

2143     #
2144     # If the user set CLOSED_BRINGOVER_WS and didn't set CLOSED_IS_PRESENT
2145     # to "no," then we'll want to initialise the closed repository
2146     #
2147     # We use $orig_closed_is_present instead of $CLOSED_IS_PRESENT,
2148     # because for newly-created source trees, the latter will be "no"
2149     # until after the bringover completes.
2150     #
2151     if [[ "$orig_closed_is_present" != "no" && \
2152         -n "$CLOSED_BRINGOVER_WS" && \
2153         ! -d $CODEMGR_WS/usr/closed/.hg ]]; then
2154         staffer mkdir -p $CODEMGR_WS/usr/closed
2155         staffer hg init $CODEMGR_WS/usr/closed
2156         staffer echo "[paths]" > $CODEMGR_WS/usr/closed/.hg/hgrc
2157         staffer echo "default=$CLOSED_BRINGOVER_WS" >> $CODEMGR_WS/usr/c/
2158         touch $TMPDIR/new_closed
2159         export CLOSED_IS_PRESENT=yes
2160     fi

2068     typeset -x HGMERGE="/bin/false"

2070     #
2071     # If the user has changes, regardless of whether those changes are
2072     # committed, and regardless of whether those changes conflict, then
2073     # we'll attempt to merge them either implicitly (uncommitted) or
2074     # explicitly (committed).
2075     #
2076     # These are the messages we'll use to help clarify mercurial output
2077     # in those cases.
2078     #
2079     typeset mergefailmsg="\
2080 ***\n\
2081 *** nightly was unable to automatically merge your changes. You should\n\
2082 *** redo the full merge manually, following the steps outlined by mercurial\n\
2083 *** above, then restart nightly.\n\
2084 ***\n"
2085     typeset mergepassmsg="\
2086 ***\n\
2087 *** nightly successfully merged your changes. This means that your working\n\
2088 *** directory has been updated, but those changes are not yet committed.\n\
2089 *** After nightly completes, you should validate the results of the merge.\n\
2090 *** then use hg commit manually.\n\
2091 ***\n"

```

```

2093 #
2094 # For each repository in turn:
2095 #
2096 # 1. Do the pull.  If this fails, dump the output and bail out.
2097 #
2098 # 2. If the pull resulted in an extra head, do an explicit merge.
2099 #   If this fails, dump the output and bail out.
2100 #
2101 # Because we can't rely on Mercurial to exit with a failure code
2102 # when a merge fails (Mercurial issue #186), we must grep the
2103 # output of pull/merge to check for attempted and/or failed merges.
2104 #
2105 # 3. If a merge failed, set the message and fail the bringover.
2106 #
2107 # 4. Otherwise, if a merge succeeded, set the message
2108 #
2109 # 5. Dump the output, and any message from step 3 or 4.
2110 #

2112 typeset HG_SOURCE=$BRINGOVER_WS
2113 if [ ! -f $TMPDIR/new_repository ]; then
2114   HG_SOURCE=$TMPDIR/open_bundle.hg
2115   staffer hg --cwd $CODEMGR_WS incoming --bundle $HG_SOURCE \
2116     -v $BRINGOVER_WS > $TMPDIR/incoming_open.out

2118 #
2119 # If there are no incoming changesets, then incoming will
2120 # fail, and there will be no bundle file.  Reset the source,
2121 # to allow the remaining logic to complete with no false
2122 # negatives.  (Unlike incoming, pull will return success
2123 # for the no-change case.)
2124 #
2125 if (( $? != 0 )); then
2126   HG_SOURCE=$BRINGOVER_WS
2127 fi
2128 fi

2130 staffer hg --cwd $CODEMGR_WS pull -u $HG_SOURCE \
2131   > $TMPDIR/pull_open.out 2>&1
2132 if (( $? != 0 )); then
2133   printf "%s: pull failed as follows:\n\n" "$CODEMGR_WS"
2134   cat $TMPDIR/pull_open.out
2135   if grep "^merging.*failed" $TMPDIR/pull_open.out > /dev/null 2>&
2136     printf "$mergefailmsg"
2137 fi
2138 touch $TMPDIR/bringover_failed
2139 return
2140 fi

2142 if grep "not updating" $TMPDIR/pull_open.out > /dev/null 2>&1; then
2143   staffer hg --cwd $CODEMGR_WS merge \
2144     >> $TMPDIR/pull_open.out 2>&1
2145   if (( $? != 0 )); then
2146     printf "%s: merge failed as follows:\n\n" \
2147       "$CODEMGR_WS"
2148     cat $TMPDIR/pull_open.out
2149     if grep "^merging.*failed" $TMPDIR/pull_open.out \
2150       > /dev/null 2>&1; then
2151       printf "$mergefailmsg"
2152     fi
2153     touch $TMPDIR/bringover_failed
2154     return
2155 fi
2156 fi

```

```

2158 printf "updated %s with the following results:\n" "$CODEMGR_WS"
2159 cat $TMPDIR/pull_open.out
2160 if grep "^merging" $TMPDIR/pull_open.out >/dev/null 2>&1; then
2161   printf "$mergepassmsg"
2162 fi
2163 printf "\n"

2165 #
2166 # We only want to update usr/closed if it exists, and we haven't been
2167 # told not to via $CLOSED_IS_PRESENT, and we actually know where to
2168 # pull from ($CLOSED_BRINGOVER_WS).
2169 #
2170 if [[ $CLOSED_IS_PRESENT = yes && \
2171     -d $CODEMGR_WS/usr/closed/.hg && \
2172     -n $CLOSED_BRINGOVER_WS ]]; then

2174   HG_SOURCE=$CLOSED_BRINGOVER_WS
2175   if [ ! -f $TMPDIR/new_closed ]; then
2176     HG_SOURCE=$TMPDIR/closed_bundle.hg
2177     staffer hg --cwd $CODEMGR_WS/usr/closed incoming \
2178       --bundle $HG_SOURCE -v $CLOSED_BRINGOVER_WS \
2179       > $TMPDIR/incoming_closed.out

2181 #
2182 # If there are no incoming changesets, then incoming will
2183 # fail, and there will be no bundle file.  Reset the sou
2184 # to allow the remaining logic to complete with no false
2185 # negatives.  (Unlike incoming, pull will return success
2186 # for the no-change case.)
2187 #
2188 if (( $? != 0 )); then
2189   HG_SOURCE=$CLOSED_BRINGOVER_WS
2190 fi
2191 fi

2193 staffer hg --cwd $CODEMGR_WS/usr/closed pull -u \
2194   $HG_SOURCE > $TMPDIR/pull_closed.out 2>&1
2195 if (( $? != 0 )); then
2196   printf "closed pull failed as follows:\n\n"
2197   cat $TMPDIR/pull_closed.out
2198   if grep "^merging.*failed" $TMPDIR/pull_closed.out \
2199     > /dev/null 2>&1; then
2200     printf "$mergefailmsg"
2201 fi
2202 touch $TMPDIR/bringover_failed
2203 return
2204 fi

2206 if grep "not updating" $TMPDIR/pull_closed.out > /dev/null 2>&1;
2207 staffer hg --cwd $CODEMGR_WS/usr/closed merge \
2208   >> $TMPDIR/pull_closed.out 2>&1
2209 if (( $? != 0 )); then
2210   printf "closed merge failed as follows:\n\n"
2211   cat $TMPDIR/pull_closed.out
2212   if grep "^merging.*failed" $TMPDIR/pull_closed.o
2213     printf "$mergefailmsg"
2214 fi
2215 touch $TMPDIR/bringover_failed
2216 return
2217 fi
2218 fi

2220 printf "updated %s with the following results:\n" \
2221   "$CODEMGR_WS/usr/closed"
2222 cat $TMPDIR/pull_closed.out
2223 if grep "^merging" $TMPDIR/pull_closed.out > /dev/null 2>&1; the

```

```

2318         printf "$mergespassmsg"
2319     fi
2320 fi

2322 #
23166 # Per-changeset output is neither useful nor manageable for a
23167 # newly-created repository.
23168 #
23169 if [ -f $TMPDIR/new_repository ]; then
23170     return
23171 fi

23173 printf "\nadded the following changesets to open repository:\n"
23174 cat $TMPDIR/incoming_open.out

23176 #
23177 # The closed repository could have been newly created, even though
23178 # the open one previously existed...
23179 #
23180 if [ -f $TMPDIR/new_closed ]; then
23181     return
23182 fi

23184 if [ -f $TMPDIR/incoming_closed.out ]; then
23185     printf "\nadded the following changesets to closed repository:\n"
23186     cat $TMPDIR/incoming_closed.out
23187 fi
23188 }

```

unchanged portion omitted

```

2225 #
2226 #     Decide whether to bringover to the codemgr workspace
2227 #
2228 if [ "$n_FLAG" = "n" ]; then
2229     PARENT_SCM_TYPE=$(parent_wstype)

2231     if [[ $SCM_TYPE != none && $SCM_TYPE != $PARENT_SCM_TYPE ]]; then
2232         echo "cannot bringover from $PARENT_SCM_TYPE to $SCM_TYPE, " \
2233             "quitting at `date`." | tee -a $mail_msg_file >> $LOGFILE
2234     fi
2235 fi

2237 run_hook PRE_BRINGOVER

2239 echo "\n=== bringover to $CODEMGR_WS at `date` ===\n" >> $LOGFILE
2240 echo "\n=== BRINGOVER LOG ===\n" >> $mail_msg_file

2242 eval "bringover_${PARENT_SCM_TYPE}" 2>&1 |
2243     tee -a $mail_msg_file >> $LOGFILE

2245 if [ -f $TMPDIR/bringover_failed ]; then
2246     rm -f $TMPDIR/bringover_failed
2247     build_ok=n
2248     echo "trouble with bringover, quitting at `date`." |
2249         tee -a $mail_msg_file >> $LOGFILE
2250     exit 1
2251 fi

2253 #
2254 # It's possible that we used the bringover above to create
2255 # $CODEMGR_WS. If so, then SCM_TYPE was previously "none,"
2256 # but should now be the same as $BRINGOVER_WS.
2257 #
2258 [[ $SCM_TYPE = none ]] && SCM_TYPE=$PARENT_SCM_TYPE

2260 run_hook POST_BRINGOVER

```

```

2419 #
2420 # Possible transition from pre-split workspace to split
2421 # workspace. See if the bringover changed anything.
2422 #
2423 CLOSED_IS_PRESENT="$orig_closed_is_present"
2424 check_closed_tree

2264 else
2265     echo "\n=== No bringover to $CODEMGR_WS ===\n" >> $LOGFILE
2266 fi

2268 if [[ "$O_FLAG" = y ]]; then
2430 if [[ "$O_FLAG" = y && "$CLOSED_IS_PRESENT" != "yes" ]]; then
2269     build_ok=n
2270     echo "OpenSolaris binary deliverables need usr/closed." \
2271         | tee -a "$mail_msg_file" >> $LOGFILE
2272     exit 1
2273 fi

2275 # Safeguards
2276 [[ -v CODEMGR_WS ]] || fatal_error "Error: Variable CODEMGR_WS not set."
2277 [[ -d "${CODEMGR_WS}" ]] || fatal_error "Error: ${CODEMGR_WS} is not a directory"
2278 [[ -f "${CODEMGR_WS}/usr/src/Makefile" ]] || fatal_error "Error: ${CODEMGR_WS}/u

2280 echo "\n=== Build environment ===\n" | tee -a $build_envIRON_file >> $LOGFILE

2282 # System
2283 whence uname | tee -a $build_envIRON_file >> $LOGFILE
2284 uname -a 2>&1 | tee -a $build_envIRON_file >> $LOGFILE
2285 echo | tee -a $build_envIRON_file >> $LOGFILE

2287 # make
2288 whence $MAKE | tee -a $build_envIRON_file >> $LOGFILE
2289 $MAKE -v | tee -a $build_envIRON_file >> $LOGFILE
2290 echo "number of concurrent jobs = $DMAKE_MAX_JOBS" |
2291     tee -a $build_envIRON_file >> $LOGFILE

2293 #
2294 # Report the compiler versions.
2295 #

2297 if [[ ! -f $SRC/Makefile ]]; then
2298     build_ok=n
2299     echo "\nUnable to find \"Makefile\" in $SRC." | \
2300         tee -a $build_envIRON_file >> $LOGFILE
2301     exit 1
2302 fi

2304 ( cd $SRC
2305     for target in cc-version cc64-version java-version; do
2306         echo
2307         #
2308         # Put statefile somewhere we know we can write to rather than trip
2309         # over a read-only $srcroot.
2310         #
2311         rm -f $TMPDIR/make-state
2312         export SRC
2313         if $MAKE -K $TMPDIR/make-state -e $target 2>/dev/null; then
2314             continue
2315         fi
2316         touch $TMPDIR/nocompiler
2317     done
2318     echo
2319 ) | tee -a $build_envIRON_file >> $LOGFILE

```

new/usr/src/tools/scripts/nightly.sh

15

```

2321 if [ -f $TMPDIR/nocompiler ]; then
2322     rm -f $TMPDIR/nocompiler
2323     build_ok=n
2324     echo "Aborting due to missing compiler." |
2325         tee -a $build_environ_file >> $LOGFILE
2326     exit 1
2327 fi

2329 # as
2330 whence ld | tee -a $build_environ_file >> $LOGFILE
2331 as -V 2>&l | head -1 | tee -a $build_environ_file >> $LOGFILE
2332 echo | tee -a $build_environ_file >> $LOGFILE

2334 # Check that we're running a capable link-editor
2335 whence ld | tee -a $build_environ_file >> $LOGFILE
2336 LDVER='ld -V 2>&l'
2337 echo $LDVER | tee -a $build_environ_file >> $LOGFILE
2338 LDVER='echo $LDVER | sed -e "s/.*-1\\.\\([0-9]*\\).*/\\1/"'
2339 if [ `expr $LDVER \\< 422` -eq 1 ]; then
2340     echo "The link-editor needs to be at version 422 or higher to build" | \
2341         tee -a $build_environ_file >> $LOGFILE
2342     echo "the latest stuff. Hope your build works." | \
2343         tee -a $build_environ_file >> $LOGFILE
2344 fi

2346 #
2347 # Build and use the workspace's tools if requested
2348 #
2349 if [[ "$t_FLAG" = "y" || "$O_FLAG" = y ]]; then
2350     set_non_debug_build_flags

2352     build_tools ${TOOLS_PROTO}
2353     if [[ $? != 0 && "$t_FLAG" = y ]]; then
2354         use_tools $TOOLS_PROTO
2355     fi
2356 fi

2358 #
2359 # copy ihv proto area in addition to the build itself
2360 #
2361 if [ "$X_FLAG" = "y" ]; then
2362     copy_ihv_proto
2363 fi

2365 if [ "$i_FLAG" = "y" -a "$SH_FLAG" = "y" ]; then
2366     echo "\n=== NOT Building base OS-Net source ===\n" | \
2367         tee -a $LOGFILE >> $mail_msg_file
2368 else
2369     # timestamp the start of the normal build; the findunref tool uses it.
2370     touch $SRC/.build.tstamp

2372     normal_build
2373 fi

2375 #
2376 # Generate the THIRDPARTYLICENSE files if needed. This is done after
2377 # the build, so that dynamically-created license files are there.
2378 # It's done before findunref to help identify license files that need
2379 # to be added to tools/opensolaris/license-list.
2380 #
2381 if [ "$O_FLAG" = y -a "$build_ok" = y ]; then
2382     echo "\n=== Generating THIRDPARTYLICENSE files ===\n" |
2383         tee -a $mail_msg_file >> $LOGFILE"

2385     if [ -d $ROOT/licenses/usr ]; then
2386         ( cd $ROOT/licenses ; \

```

new/usr/src/tools/scripts/nightly.sh

16

```

2387         mktpl $SRC/pkg/license-list ) >> "$LOGFILE" 2>&l
2388         if (( $? != 0 )) ; then
2389             echo "Couldn't create THIRDPARTYLICENSE files" |
2390                 tee -a $mail_msg_file >> "$LOGFILE"
2391         fi
2392     else
2393         echo "No licenses found under $ROOT/licenses" |
2394             tee -a $mail_msg_file >> "$LOGFILE"
2395     fi
2396 fi

2398 ORIG_SRC=$SRC
2399 BINARCHIVE=${CODEMGR_WS}/bin-{$MACH}.cpio.Z

2401 if [ "$SE_FLAG" = "y" -o "$SD_FLAG" = "y" -o "$SH_FLAG" = "y" ]; then
2402     save_binaries
2403 fi

2406 # EXPORT_SRC comes after CRYPT_SRC since a domestic build will need
2407 # $SRC pointing to the export_source usr/src.

2409 if [ "$SE_FLAG" = "y" -o "$SD_FLAG" = "y" -o "$SH_FLAG" = "y" ]; then
2410     if [ "$SD_FLAG" = "y" -a $build_ok = y ]; then
2411         set_up_source_build ${CODEMGR_WS} ${CRYPT_SRC} CRYPT_SRC
2412     fi

2414     if [ $build_ok = y ]; then
2415         set_up_source_build ${CODEMGR_WS} ${EXPORT_SRC} EXPORT_SRC
2416     fi
2417 fi

2419 if [ "$SD_FLAG" = "y" -a $build_ok = y ]; then
2420     # drop the crypt files in place.
2421     cd ${EXPORT_SRC}
2422     echo "\nexttracting crypt_files.cpio.Z onto export_source.\n" \
2423         >> $LOGFILE
2424     zcat ${CODEMGR_WS}/crypt_files.cpio.Z | \
2425         cpio -idmucvB 2>/dev/null >> $LOGFILE
2426     if [ "$?" = "0" ]; then
2427         echo "\n=== DOMESTIC extraction succeeded ===\n" \
2428             >> $mail_msg_file
2429     else
2430         echo "\n=== DOMESTIC extraction failed ===\n" \
2431             >> $mail_msg_file
2432     fi

2434 fi

2436 if [ "$SO_FLAG" = "y" -a "$build_ok" = y ]; then
2437     #
2438     # Copy the open sources into their own tree.
2439     # If copy_source fails, it will have already generated an
2440     # error message and set build_ok=n, so we don't need to worry
2441     # about that here.
2442     #
2443     copy_source $CODEMGR_WS $OPEN_SRCDIR OPEN_SOURCE usr/src
2444 fi

2446 if [ "$SO_FLAG" = "y" -a "$build_ok" = y ]; then
2447     SRC=$OPEN_SRCDIR/usr/src
2448     export CLOSED_IS_PRESENT=no
2449 fi

2450 if is_source_build && [ $build_ok = y ]; then
2451     # remove proto area(s) here, since we don't clobber

```

```

2452 rm -rf `allprotos`
2453 if [ "$t_FLAG" = "y" ]; then
2454     set_non_debug_build_flags
2455     ORIG_TOOLS=$TOOLS
2456     #
2457     # SRC was set earlier to point to the source build
2458     # source tree (e.g., $EXPORT_SRC).
2459     #
2460     TOOLS=${SRC}/tools
2461     TOOLS_PROTO=${TOOLS}/${TOOLS_PROTO_REL}; export TOOLS_PROTO
2462     build_tools ${TOOLS_PROTO}
2463     if [[ $? != 0 ]]; then
2464         use_tools ${TOOLS_PROTO}
2465     fi
2466 fi

2468 normal_build
2469 fi

2471 #
2472 # There are several checks that need to look at the proto area, but
2473 # they only need to look at one, and they don't care whether it's
2474 # DEBUG or non-DEBUG.
2475 #
2476 if [[ "$MULTI_PROTO" = yes && "$D_FLAG" = n ]]; then
2477     checkroot=$ROOT-nd
2478 else
2479     checkroot=$ROOT
2480 fi

2482 if [ "$build_ok" = "y" ]; then
2483     echo "\n=== Creating protolist system file at `date` ===" \
2484         >> $LOGFILE
2485     protolist $checkroot > $ATLOG/proto_list_${MACH}
2486     echo "=== protolist system file created at `date` ===\n" \
2487         >> $LOGFILE

2489     if [ "$N_FLAG" != "y" ]; then

2491         E1=
2492         f1=
2493         if [ -d "$SRC/pkgdefs" ]; then
2494             f1="$SRC/pkgdefs/etc/exception_list_${MACH}"
2495             if [ "$X_FLAG" = "y" ]; then
2496                 f1="$f1 $IA32_IHV_WS/usr/src/pkgdefs/etc/excepti
2497             fi
2498         fi

2500     for f in $f1; do
2501         if [ -f "$f" ]; then
2502             E1="$E1 -e $f"
2503         fi
2504     done

2506     E2=
2507     f2=
2508     if [ -d "$SRC/pkg" ]; then
2509         f2="$f2 exceptions/packaging"
2510     fi

2512     for f in $f2; do
2513         if [ -f "$f" ]; then
2514             E2="$E2 -e $f"
2515         fi
2516     done

```

```

2518     if [ -f "$REF_PROTO_LIST" ]; then
2519         #
2520         # For builds that copy the IHV proto area (-X), add the
2521         # IHV proto list to the reference list if the reference
2522         # was built without -X.
2523         #
2524         # For builds that don't copy the IHV proto area, add the
2525         # IHV proto list to the build's proto list if the
2526         # reference was built with -X.
2527         #
2528         # Use the presence of the first file entry of the cached
2529         # IHV proto list in the reference list to determine
2530         # whether it was built with -X or not.
2531         #
2532         IHV_REF_PROTO_LIST=$SRC/pkg/proto_list_ihv_${MACH}
2533         grepfor=$(nawk 's1 == "f" { print $2; exit }' \
2534             $IHV_REF_PROTO_LIST 2> /dev/null)
2535         if [ $? = 0 -a -n "$grepfor" ]; then
2536             if [ "$X_FLAG" = "y" ]; then
2537                 grep -w "$grepfor" \
2538                     $REF_PROTO_LIST > /dev/null
2539                 if [ ! "$?" = "0" ]; then
2540                     REF_IHV_PROTO="-d $IHV_REF_PROTO
2541                 fi
2542             else
2543                 grep -w "$grepfor" \
2544                     $REF_PROTO_LIST > /dev/null
2545                 if [ "$?" = "0" ]; then
2546                     IHV_PROTO_LIST="$IHV_REF_PROTO_L
2547                 fi
2548             fi
2549         fi
2550     fi
2551 fi

2553 if [ "$N_FLAG" != "y" -a -f $SRC/pkgdefs/Makefile ]; then
2554     echo "\n=== Impact on SVr4 packages ===\n" >> $mail_msg_file
2555     #
2556     # Compare the build's proto list with current package
2557     # definitions to audit the quality of package
2558     # definitions and makefile install targets. Use the
2559     # current exception list.
2560     #
2561     PKGDEFS_LIST=""
2562     for d in $abssrcdirs; do
2563         if [ -d $d/pkgdefs ]; then
2564             PKGDEFS_LIST="$PKGDEFS_LIST -d $d/pkgdefs"
2565         fi
2566     done
2567     if [ "$X_FLAG" = "y" -a \
2568         -d $IA32_IHV_WS/usr/src/pkgdefs ]; then
2569         PKGDEFS_LIST="$PKGDEFS_LIST -d $IA32_IHV_WS/usr/src/pkgd
2570     fi
2571     $PROTOCMPTRSE \
2572         "Files missing from the proto area:" \
2573         "Files missing from packages:" \
2574         "Inconsistencies between pkgdefs and proto area:" \
2575         ${E1} \
2576         ${PKGDEFS_LIST} \
2577         $ATLOG/proto_list_${MACH} \
2578         >> $mail_msg_file
2579 fi

2581 if [ "$N_FLAG" != "y" -a -d $SRC/pkg ]; then
2582     echo "\n=== Validating manifests against proto area ===\n" \
2583         >> $mail_msg_file

```

```

2584         ( cd $SRC/pkg ; $MAKE -e protocmp ROOT="$checkroot" ) \
2585         >> $mail_msg_file

2587     fi

2589     if [ "$N_FLAG" != "y" -a -f "$REF_PROTO_LIST" ]; then
2590         echo "\n=== Impact on proto area ===\n" >> $mail_msg_file
2591         if [ -n "$E2" ]; then
2592             ELIST=$E2
2593         else
2594             ELIST=$E1
2595         fi
2596         $PROTOCMPTRSE \
2597             "Files in yesterday's proto area, but not today's:" \
2598             "Files in today's proto area, but not yesterday's:" \
2599             "Files that changed between yesterday and today:" \
2600             ${ELIST} \
2601             -d $REF_PROTO_LIST \
2602             $REF_IHV_PROTO \
2603             $ATLOG/proto_list_${MACH} \
2604             $IHV_PROTO_LIST \
2605             >> $mail_msg_file
2606     fi
2607 fi

2609 if [ "$U_FLAG" = "y" -a "$build_ok" = "y" ]; then
2610     staffer cp $ATLOG/proto_list_${MACH} \
2611             $PARENT_WS/usr/src/proto_list_${MACH}
2612 fi

2614 # Update parent proto area if necessary. This is done now
2615 # so that the proto area has either DEBUG or non-DEBUG kernels.
2616 # Note that this clears out the lock file, so we can dispense with
2617 # the variable now.
2618 if [ "$SU_FLAG" = "y" -a "$build_ok" = "y" ]; then
2619     echo "\n=== Copying proto area to $NIGHTLY_PARENT_ROOT ===\n" | \
2620     tee -a $LOGFILE >> $mail_msg_file
2621     rm -rf $NIGHTLY_PARENT_ROOT/*
2622     unset Ulockfile
2623     mkdir -p $NIGHTLY_PARENT_ROOT
2624     if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
2625         ( cd $ROOT; tar cf - . |
2626           ( cd $NIGHTLY_PARENT_ROOT; umask 0; tar xpf - ) ) 2>&1 |
2627         tee -a $mail_msg_file >> $LOGFILE
2628     fi
2629     if [[ "$MULTI_PROTO" = yes && "$F_FLAG" = n ]]; then
2630         rm -rf $NIGHTLY_PARENT_ROOT-nd/*
2631         mkdir -p $NIGHTLY_PARENT_ROOT-nd
2632         cd $ROOT-nd
2633         ( tar cf - . |
2634           ( cd $NIGHTLY_PARENT_ROOT-nd; umask 0; tar xpf - ) ) 2>&1 |
2635         tee -a $mail_msg_file >> $LOGFILE
2636     fi
2637     if [ -n "${NIGHTLY_PARENT_TOOLS_ROOT}" ]; then
2638         echo "\n=== Copying tools proto area to $NIGHTLY_PARENT_TOOLS_ROOT"
2639         tee -a $LOGFILE >> $mail_msg_file
2640         rm -rf $NIGHTLY_PARENT_TOOLS_ROOT/*
2641         mkdir -p $NIGHTLY_PARENT_TOOLS_ROOT
2642         if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
2643             ( cd $TOOLS_PROTO; tar cf - . |
2644               ( cd $NIGHTLY_PARENT_TOOLS_ROOT;
2645                 umask 0; tar xpf - ) ) 2>&1 |
2646             tee -a $mail_msg_file >> $LOGFILE
2647         fi
2648     fi
2649 fi

```

```

2651 #
2652 # ELF verification: ABI (-A) and runtime (-r) checks
2653 #
2654 if [[ ($build_ok = y) && ( ($A_FLAG = y) || ($r_FLAG = y) ) ]]; then
2655     # Directory ELF-data.$MACH holds the files produced by these tests.
2656     elf_dkdir=$SRC/ELF-data.$MACH

2658     # If there is a previous ELF-data backup directory, remove it. Then,
2659     # rotate current ELF-data directory into its place and create a new
2660     # empty directory
2661     rm -rf $elf_dkdir.ref
2662     if [[ -d $elf_dkdir ]]; then
2663         mv $elf_dkdir $elf_dkdir.ref
2664     fi
2665     mkdir -p $elf_dkdir

2667     # Call find_elf to produce a list of the ELF objects in the proto area.
2668     # This list is passed to check_rtime and interface_check, preventing
2669     # them from separately calling find_elf to do the same work twice.
2670     find_elf -fr $checkroot > $elf_dkdir/object_list

2672     if [[ $A_FLAG = y ]]; then
2673         echo "\n=== Check versioning and ABI information ===\n" | \
2674         tee -a $LOGFILE >> $mail_msg_file

2676         # Produce interface description for the proto. Report errors.
2677         interface_check -o -w $elf_dkdir -f object_list \
2678             -i interface -E interface.err
2679         if [[ -s $elf_dkdir/interface.err ]]; then
2680             tee -a $LOGFILE < $elf_dkdir/interface.err \
2681             >> $mail_msg_file
2682         fi

2684         # If ELF_DATA_BASELINE_DIR is defined, compare the new interface
2685         # description file to that from the baseline gate. Issue a
2686         # warning if the baseline is not present, and keep going.
2687         if [[ "$ELF_DATA_BASELINE_DIR" != '' ]]; then
2688             base_ifile="$ELF_DATA_BASELINE_DIR/interface"

2690             echo "\n=== Compare versioning and ABI information" \
2691                 "to baseline ===\n" | \
2692             tee -a $LOGFILE >> $mail_msg_file
2693             echo "Baseline: $base_ifile\n" >> $LOGFILE

2695             if [[ -f $base_ifile ]]; then
2696                 interface_cmp -d -o $base_ifile \
2697                     $elf_dkdir/interface > $elf_dkdir/interface.cm
2698                 if [[ -s $elf_dkdir/interface.cm ]]; then
2699                     echo | tee -a $LOGFILE >> $mail_msg_file
2700                     tee -a $LOGFILE < \
2701                         $elf_dkdir/interface.cm \
2702                         >> $mail_msg_file
2703                 fi
2704             else
2705                 echo "baseline not available. comparison" \
2706                     "skipped" | \
2707                 tee -a $LOGFILE >> $mail_msg_file
2708             fi

2710         fi
2711     fi

2713     if [[ $r_FLAG = y ]]; then
2714         echo "\n=== Check ELF runtime attributes ===\n" | \
2715         tee -a $LOGFILE >> $mail_msg_file

```

```

2717     # If we're doing a DEBUG build the proto area will be left
2718     # with debuggable objects, thus don't assert -s.
2719     if [[ $D_FLAG = y ]]; then
2720         rtime_sflag=""
2721     else
2722         rtime_sflag="-s"
2723     fi
2724     check_rtime -i -m -v $rtime_sflag -o -w $self_ddir \
2725         -D object_list -f object_list -E runtime.err \
2726         -I runtime.attr.raw

2728     # check_rtime -I output needs to be sorted in order to
2729     # compare it to that from previous builds.
2730     sort $self_ddir/runtime.attr.raw > $self_ddir/runtime.attr
2731     rm $self_ddir/runtime.attr.raw

2733     # Report errors
2734     if [[ -s $self_ddir/runtime.err ]]; then
2735         tee -a $LOGFILE < $self_ddir/runtime.err \
2736             >> $mail_msg_file
2737     fi

2739     # If there is an ELF-data directory from a previous build,
2740     # then diff the attr files. These files contain information
2741     # about dependencies, versioning, and runpaths. There is some
2742     # overlap with the ABI checking done above, but this also
2743     # flushes out non-ABI interface differences along with the
2744     # other information.
2745     echo "\n=== Diff ELF runtime attributes" \
2746         "(since last build) ===\n" | \
2747         tee -a $LOGFILE >> $mail_msg_file >> $mail_msg_file

2749     if [[ -f $self_ddir.ref/runtime.attr ]]; then
2750         diff $self_ddir.ref/runtime.attr \
2751             $self_ddir/runtime.attr \
2752             >> $mail_msg_file
2753     fi
2754 fi

2756 # If -u set, copy contents of ELF-data.$MACH to the parent workspace.
2757 if [[ "$u_FLAG" = "y" ]]; then
2758     p_elf_ddir=$PARENT_WS/usr/ELF-data.$MACH

2760     # If parent lacks the ELF-data.$MACH directory, create it
2761     if [[ ! -d $p_elf_ddir ]]; then
2762         staffer mkdir -p $p_elf_ddir
2763     fi

2765     # These files are used asynchronously by other builds for ABI
2766     # verification, as above for the -A option. As such, we require
2767     # the file replacement to be atomic. Copy the data to a temp
2768     # file in the same filesystem and then rename into place.
2769     (
2770         cd $self_ddir
2771         for elf_dfile in *; do
2772             staffer cp $elf_dfile \
2773                 ${p_elf_ddir}/${elf_dfile}.new
2774             staffer mv -f ${p_elf_ddir}/${elf_dfile}.new \
2775                 ${p_elf_ddir}/${elf_dfile}
2776         done
2777     )
2778 fi
2779 fi

2781 # DEBUG lint of kernel begins

```

```

2783 if [ "$i_CMD_LINE_FLAG" = "n" -a "$l_FLAG" = "y" ]; then
2784     if [ "$LINTDIRS" = "" ]; then
2785         # LINTDIRS="$SRC/uts y $SRC/stand y $SRC/psm y"
2786         LINTDIRS="$SRC y"
2787     fi
2788     set $LINTDIRS
2789     while [ $# -gt 0 ]; do
2790         dolint $1 $2; shift; shift
2791     done
2792 else
2793     echo "\n=== No '$MAKE lint' ===\n" >> $LOGFILE
2794 fi

2796 # "make check" begins

2798 if [ "$i_CMD_LINE_FLAG" = "n" -a "$C_FLAG" = "y" ]; then
2799     # remove old check.out
2800     rm -f $SRC/check.out

2802     rm -f $SRC/check-${MACH}.out
2803     cd $SRC
2804     $MAKE -ek check ROOT="$checkroot" 2>&1 | tee -a $SRC/check-${MACH}.out \
2805         >> $LOGFILE
2806     echo "\n=== cstyle/hdrchk errors ===\n" >> $mail_msg_file

2808     grep ":" $SRC/check-${MACH}.out |
2809         egrep -v "Ignoring unknown host" | \
2810         sort | uniq >> $mail_msg_file
2811 else
2812     echo "\n=== No '$MAKE check' ===\n" >> $LOGFILE
2813 fi

2815 echo "\n=== Find core files ===\n" | \
2816     tee -a $LOGFILE >> $mail_msg_file

2818 find $absrsrcdirs -name core -a -type f -exec file {} \; | \
2819     tee -a $LOGFILE >> $mail_msg_file

2821 if [ "$f_FLAG" = "y" -a "$build_ok" = "y" ]; then
2822     echo "\n=== Diff unreferenced files (since last build) ===\n" \
2823         | tee -a $LOGFILE >> $mail_msg_file
2824     rm -f $SRC/unref-${MACH}.ref
2825     if [ -f $SRC/unref-${MACH}.out ]; then
2826         mv $SRC/unref-${MACH}.out $SRC/unref-${MACH}.ref
2827     fi

2829     findunref -S $SCM_TYPE -t $SRC/.build.tstamp -s usr $CODEMGR_WS \
2830         ${TOOLS}/findunref/exception_list 2>> $mail_msg_file | \
2831         sort > $SRC/unref-${MACH}.out

2833     if [ ! -f $SRC/unref-${MACH}.ref ]; then
2834         cp $SRC/unref-${MACH}.out $SRC/unref-${MACH}.ref
2835     fi

2837     diff $SRC/unref-${MACH}.ref $SRC/unref-${MACH}.out >> $mail_msg_file
2838 fi

2840 #
2841 # Generate the OpenSolaris deliverables if requested. Some of these
2842 # steps need to come after findunref and are commented below.
2843 #

2845 # If we are doing an OpenSolaris _source_ build (-S 0) then we do
2846 # not have usr/closed available to us to generate closedbins from,
2847 # so skip this part.

```



```

2848 if [ "$SO_FLAG" = n -a "$O_FLAG" = y -a "$build_ok" = y ]; then
2849     echo "\n==== Generating OpenSolaris tarballs ==== \n" | \
2850         tee -a $mail_msg_file >> $LOGFILE
2852     cd $CODEMGR_WS
2854     #
2855     # This step grovels through the package manifests, so it
2856     # must come after findunref.
2857     #
2858     # We assume no DEBUG vs non-DEBUG package content variation
2859     # here; if that changes, then the "make all" in $SRC/pkg will
2860     # need to be moved into the conditionals and repeated for each
2861     # different build.
2862     #
2863     echo "Generating closed binaries tarball(s)..." >> $LOGFILE
2864     closed_basename=on-closed-bins
2865     if [ "$D_FLAG" = y ]; then
2866         bindrop "$closed_basename" >> "$LOGFILE" 2>&1
2867         if (( $? != 0 )) ; then
2868             echo "Couldn't create DEBUG closed binaries." |
2869             tee -a $mail_msg_file >> $LOGFILE
2870             build_ok=n
2871         fi
2872     fi
2873     if [ "$F_FLAG" = n ]; then
2874         bindrop -n "$closed_basename-nd" >> "$LOGFILE" 2>&1
2875         if (( $? != 0 )) ; then
2876             echo "Couldn't create non-DEBUG closed binaries." |
2877             tee -a $mail_msg_file >> $LOGFILE
2878             build_ok=n
2879         fi
2880     fi
2882     echo "Generating README.opensolaris..." >> $LOGFILE
2883     cat $SRC/tools/opensolaris/README.opensolaris.tmpl | \
2884     mkreadme_osol $CODEMGR_WS/README.opensolaris >> $LOGFILE 2>&1
2885     if (( $? != 0 )) ; then
2886         echo "Couldn't create README.opensolaris." |
2887         tee -a $mail_msg_file >> $LOGFILE
2888         build_ok=n
2889     fi
2890 fi
2892 # Verify that the usual lists of files, such as exception lists,
2893 # contain only valid references to files.  If the build has failed,
2894 # then don't check the proto area.
2895 CHECK_PATHS=${CHECK_PATHS:-y}
2896 if [ "$CHECK_PATHS" = y -a "$N_FLAG" != y ]; then
2897     echo "\n==== Check lists of files ==== \n" | tee -a $LOGFILE \
2898         >> $mail_msg_file
2899     arg=-b
2900     [ "$build_ok" = y ] && arg=
2901     checkpaths $arg $checkroot 2>&1 | tee -a $LOGFILE >> $mail_msg_file
2902 fi
2904 if [ "$M_FLAG" != "y" -a "$build_ok" = y ]; then
2905     echo "\n==== Impact on file permissions ==== \n" \
2906         >> $mail_msg_file
2908     abspkgdefs=
2909     abspkg=
2910     for d in $abssrcdirs; do
2911         if [ -d "$d/pkgdefs" ]; then
2912             abspkgdefs="$abspkgdefs $d"
2913         fi

```

```

2914         if [ -d "$d/pkg" ]; then
2915             abspkg="$abspkg $d"
2916         fi
2917     done
2919     if [ -n "$abspkgdefs" ]; then
2920         pmodes -qvdp \
2921             'find $abspkgdefs -name pkginfo.tmpl -print -o \
2922             -name .del\* -prune | sed -e 's:/pkginfo.tmpl$::' | \
2923             sort -u' >> $mail_msg_file
2924     fi
2926     if [ -n "$abspkg" ]; then
2927         for d in "$abspkg"; do
2928             ( cd $d/pkg ; $MAKE -e pmodes ) >> $mail_msg_file
2929         done
2930     fi
2931 fi
2933 if [ "$w_FLAG" = "y" -a "$build_ok" = "y" ]; then
2934     if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
2935         do_wsdiff DEBUG $ROOT.prev $ROOT
2936     fi
2938     if [[ "$MULTI_PROTO" = yes && "$F_FLAG" = n ]]; then
2939         do_wsdiff non-DEBUG $ROOT-nd.prev $ROOT-nd
2940     fi
2941 fi
2943 END_DATE=`date`
2944 echo "==== Nightly $maketype build completed: $END_DATE ==== \n" | \
2945     tee -a $LOGFILE >> $build_time_file
2947 typeset -i10 hours
2948 typeset -Z2 minutes
2949 typeset -Z2 seconds
2951 elapsed_time=$SECONDS
2952 ((hours = elapsed_time / 3600 ))
2953 ((minutes = elapsed_time / 60 % 60))
2954 ((seconds = elapsed_time % 60))
2956 echo "\n==== Total build time ==== \n" | \
2957     tee -a $LOGFILE >> $build_time_file
2958 echo "\nreal    ${hours}:${minutes}:${seconds}" | \
2959     tee -a $LOGFILE >> $build_time_file
2961 if [ "$u_FLAG" = "y" -a "$f_FLAG" = "y" -a "$build_ok" = "y" ]; then
2962     staffer cp ${SRC}/unref-${MACH}.out $PARENT_WS/usr/src/
2964     #
2965     # Produce a master list of unreferenced files -- ideally, we'd
2966     # generate the master just once after all of the nightly
2967     # have finished, but there's no simple way to know when that
2968     # will be.  Instead, we assume that we're the last nightly to
2969     # finish and merge all of the unref-${MACH}.out files in
2970     # $PARENT_WS/usr/src/.  If we are in fact the final ${MACH} to
2971     # finish, then this file will be the authoritative master
2972     # list.  Otherwise, another ${MACH}'s nightly will eventually
2973     # overwrite ours with its own master, but in the meantime our
2974     # temporary "master" will be no worse than any older master
2975     # which was already on the parent.
2976     #
2978     set -- $PARENT_WS/usr/src/unref-*.out
2979     cp "$1" ${TMPDIR}/unref.merge

```

```
2980      shift
2982      for unreffile; do
2983          comm -12 ${TMPDIR}/unref.merge "$unreffile" > ${TMPDIR}/unref.$$
2984          mv ${TMPDIR}/unref.$$ ${TMPDIR}/unref.merge
2985      done
2987      staffer cp ${TMPDIR}/unref.merge $PARENT_WS/usr/src/unrefmaster.out
2988  fi
2990 #
2991 # All done save for the sweeping up.
2992 # (whichever exit we hit here will trigger the "cleanup" trap which
2993 # optionally sends mail on completion).
2994 #
2995 if [ "$build_ok" = "y" ]; then
2996     exit 0
2997 fi
2998 exit 1
```

new/usr/src/tools/scripts/ws.sh

1

```
*****
10351 Thu Aug 15 11:59:47 2013
new/usr/src/tools/scripts/ws.sh
4028 remove CLOSED_IS_PRESENT
*****
_____unchanged_portion_omitted_____

109 if [[ "$1" = "-e" ]]; then
110     setenv=true
111     shift
112 else
113     setenv=false
114 fi

116 WHICH_SCM=$(/bin/dirname $(whence $0))/which_scm
117 if [[ ! -x $WHICH_SCM ]]; then
118     WHICH_SCM=which_scm
119 fi

121 #
122 # No workspace/repository path was given, so try and detect one from our
123 # current directory we're in
124 #
125 if [[ $# -lt 1 ]]; then
126     if env CODEMGR_WS="" $WHICH_SCM | read SCM_MODE tmpwsname && \
127         [[ $SCM_MODE != unknown ]]; then
128         echo "Defaulting to $SCM_MODE repository $tmpwsname"
129     else
130         echo "usage: ws [-e] [workspace_name]" >&2
131         if $setenv; then
132             cleanup_env
133             return 1
134         else
135             exit 1
136         fi
137     fi
138 else
139     #
140     # A workspace/repository path was passed in, grab it and pop
141     # it off the stack
142     #
143     tmpwsname=$1
144     shift
145 fi

147 #
148 # This variable displays the nested activations of workspaces.
149 # This is done here to get the exact name the user entered.
150 #
151 WS_STACK="$tmpwsname $WS_STACK"; export WS_STACK

153 #
154 # Set the workspace name and unset tmpwsname (as we reuse it later)
155 #
156 wsname='echo $tmpwsname|fmtwsname'
157 unset tmpwsname

159 #
160 # Checking for CODEMGR_WSPATH
161 #
162 if [[ -n ${CODEMGR_WSPATH} ] && ( ! -d $wsname ) && \
163     ( `expr "$wsname" : "\/" = "0" ) ]]
164 then
165     ofs=$IFS
166     IFS=:
167     for i in $CODEMGR_WSPATH
```

new/usr/src/tools/scripts/ws.sh

2

```
168     do
169         if [[ -d ${i}/${wsname} ]]; then
170             wsname=${i}/${wsname}
171             break
172         fi
173     done
174     IFS=$ofs
175 fi

177 #
178 # to translate it to an absolute pathname. We need an
179 # absolute pathname in order to set CODEMGR_WS.
180 #
181 if [[ `expr "$wsname" : "\/" = "0" ] ]
182 then
183     pwd='pwd'
184     wsname="$pwd/$wsname"
185 fi

187 #
188 # Check to see if this is a valid workspace
189 #
190 if [[ ! -d $wsname ]]; then
191     echo "$wsname . . . no such directory" >&2
192     if $setenv; then
193         cleanup_env
194         return 1
195     else
196         exit 1
197     fi
198 fi

200 #
201 # This catches the case of a passed in workspace path
202 # Check which type of SCM is in use by $wsname.
203 #
204 (cd $wsname && env CODEMGR_WS="" $WHICH_SCM) | read SCM_MODE tmpwsname
205 if [[ $? != 0 || "$SCM_MODE" == unknown ]]; then
206     echo "Error: Unable to detect a supported SCM repository in $wsname"
207     if $setenv; then
208         cleanup_env
209         return 1
210     else
211         exit 1
212     fi
213 fi

215 wsname=$tmpwsname
216 CODEMGR_WS=$wsname ; export CODEMGR_WS
217 SRC=$wsname/usr/src; export SRC
218 TSRC=$wsname/usr/ontest; export TSRC

220 if [[ "$SCM_MODE" = "teamware" && -d ${wsname}/Codemgr_wsdata ]]; then
221     CM_DATA="Codemgr_wsdata"
222     wsosdir=$CODEMGR_WS/$CM_DATA/sunos
223     protofile=$wsosdir/protodefs
224 elif [[ "$SCM_MODE" = "mercurial" && -d ${wsname}/.hg ]]; then
225     CM_DATA=".hg"
226     wsosdir=$CODEMGR_WS/$CM_DATA
227     protofile=$wsosdir/org.opensolaris.protodefs
228 elif [[ "$SCM_MODE" = "git" && -d ${wsname}/.git ]]; then
229     CM_DATA=".git"
230     wsosdir=$CODEMGR_WS/$CM_DATA
231     protofile=$wsosdir/org.opensolaris.protodefs
232 else
233     echo "$wsname is not a supported workspace; type is $SCM_MODE" >&2
```

new/usr/src/tools/scripts/ws.sh

3

```
234     if $setenv; then
235         cleanup_env
236         return 1
237     else
238         exit 1
239     fi
240 fi

242 MACH=`uname -p`

244 if [[ ! -f $protofile ]]; then
245     if [[ ! -w $CODEMGR_WS/$CM_DATA ]]; then
246         #
247         # The workspace doesn't have a protodefs file and I am
248         # unable to create one. Tell user and use /tmp instead.
249         #
250         echo "Unable to create the proto defaults file ($protofile)."

252         # Just make one in /tmp
253         wsosdir=/tmp
254         protofile=$wsosdir/protodefs
255     fi

257     if [[ ! -d $wsosdir ]]; then
258         mkdir $wsosdir
259     fi

261     cat << PROTOFILE_EoF > $protofile
262 #!/bin/sh
263 #
264 # Set default proto areas for this workspace
265 # NOTE: This file was initially automatically generated.
266 #
267 # Feel free to edit this file. If this file is removed
268 # it will be rebuilt containing default values.
269 #
270 # The variable CODEMGR_WS is available to this script.
271 #
272 # PROTO1 is the first proto area searched and is typically set
273 # to a proto area associated with the workspace. The ROOT
274 # environment variable is set to the same as PROTO1. If you
275 # will be doing make installs this proto area needs to be writable.
276 #
277 # PROTO2 and PROTO3 are set to proto areas to search before the
278 # search proceeds to the local machine or the proto area specified by
279 # TERMPROTO.
280 #
281 # TERMPROTO (if specified) is the last place searched. If
282 # TERMPROTO is not specified the search will end at the local
283 # machine.
284 #

286 PROTO1=$CODEMGR_WS/proto
287 PROTOFILE_EoF
288
289     if [[ "$SCM_MODE" = "teamware" ]]; then
290         cat << PROTOFILE_EoF >> $protofile
291 if [[ -f "$CODEMGR_WS/Codemgr_wsdata/parent" ]]; then
292     #
293     # If this workspace has an codemgr parent then set PROTO2 to
294     # point to the parents proto space.
295     #
296     parent=`workspace parent $CODEMGR_WS`
297     if [[ -n $parent ]]; then
298         PROTO2=$parent/proto
299     fi
```

new/usr/src/tools/scripts/ws.sh

4

```
300 fi
301 PROTOFILE_EoF
302     elif [[ "$SCM_MODE" = "mercurial" ]]; then
303         cat << PROTOFILE_EoF >> $protofile
304 parent=`(cd $CODEMGR_WS && hg path default 2>/dev/null)`
305 if [[ \ $? -eq 0 && -n $parent ]]; then
306     [[ -n $(check_proto $parent/proto) ]] && PROTO2=$parent/proto
307 fi
308 PROTOFILE_EoF
309     fi
310 fi

312 . $protofile

314 # This means you don't have to type make -e all of the time

316 MAKEFLAGS=e; export MAKEFLAGS

318 #
319 # Set up the environment variables
320 #
321 ROOT=/proto/root_$(MACH) # default

323 ENVCPPFLAGS1=
324 ENVCPPFLAGS2=
325 ENVCPPFLAGS3=
326 ENVCPPFLAGS4=
327 ENVLDLIBS1=
328 ENVLDLIBS2=
329 ENVLDLIBS3=

331 PROTO1=`check_proto $PROTO1`
332 if [[ -n "$PROTO1" ]]; then # first proto area specified
333     ROOT=$PROTO1
334     ENVCPPFLAGS1=-I$ROOT/usr/include
335     export ENVCPPFLAGS1
336     ENVLDLIBS1="-L$ROOT/lib -L$ROOT/usr/lib"
337     export ENVLDLIBS1

339     PROTO2=`check_proto $PROTO2`
340     if [[ -n "$PROTO2" ]]; then # second proto area specified
341         ENVCPPFLAGS2=-I$PROTO2/usr/include
342         export ENVCPPFLAGS2
343         ENVLDLIBS2="-L$PROTO2/lib -L$PROTO2/usr/lib"
344         export ENVLDLIBS2

346         PROTO3=`check_proto $PROTO3`
347         if [[ -n "$PROTO3" ]]; then # third proto area specified
348             ENVCPPFLAGS3=-I$PROTO3/usr/include
349             export ENVCPPFLAGS3
350             ENVLDLIBS3="-L$PROTO3/lib -L$PROTO3/usr/lib"
351             export ENVLDLIBS3
352         fi
353     fi
354 fi

356 export ROOT

358 if [[ -n "$TERMPROTO" ]]; then # fallback area specified
359     TERMPROTO=`check_proto $TERMPROTO`
360     ENVCPPFLAGS4="-Y I,$TERMPROTO/usr/include"
361     export ENVCPPFLAGS4
362     ENVLDLIBS3="$ENVLDLIBS3 -Y P,$TERMPROTO/lib:$TERMPROTO/usr/lib"
363     export ENVLDLIBS3
364 fi
```

```

366 osbld_flag=0

368 if [[ ! -v CLOSED_IS_PRESENT ]]; then
369     if [[ -d $SRC/../closed ]]; then
370         export CLOSED_IS_PRESENT="yes"
371     else
372         export CLOSED_IS_PRESENT="no"
373     fi
374 fi

368 if [[ -z "$ONBLD_DIR" ]]; then
369     ONBLD_DIR=$(/bin/dirname $(whence $0))
370 fi

372 if ! echo ":$PATH:" | grep ":{ONBLD_DIR}:" > /dev/null; then
373     PATH="{ONBLD_DIR}:{ONBLD_DIR}/${MACH}:"${PATH}"
374     osbld_flag=1
375 fi

377 export PATH

379 if [[ -n "$PROTO2" ]]; then
380     # This should point to the parent's proto
381     PARENT_ROOT=$PROTO2
382     export PARENT_ROOT
383 else
384     # Clear it in case it's already in the env.
385     PARENT_ROOT=
386 fi
387 export ONBLD_DIR
388 export MACH

390 os_rev=`uname -r`
391 os_name=`uname -s`

393 if [[ $os_name != "SunOS" || `expr $os_rev : "5\."` != "2" ]]; then
394     #
395     # This is not a SunOS 5.x machine - something is wrong
396     #
397     echo "***WARNING: this script is meant to be run on SunOS 5.x."
398     echo "          This machine appears to be running: $os_name $os_rev"
399 fi

401 echo ""
402 echo "Workspace                : $wsname"
403 if [[ -n "$parent" ]]; then
404     echo "Workspace Parent          : $parent"
405 fi
406 echo "Proto area (\$ROOT)       : $ROOT"
407 if [[ -n "$PARENT_ROOT" ]]; then
408     echo "Parent proto area (\$PARENT_ROOT) : $PARENT_ROOT"
409 fi
410 echo "Root of source (\$SRC)      : $SRC"
411 echo "Root of test source (\$TSRC) : $TSRC"
412 if [[ $osbld_flag = "1" ]]; then
413     echo "Prepended to PATH          : $ONBLD_DIR"
414 fi
415 echo "Current directory (\$PWD)    : $wsname"
416 echo ""

418 cd $wsname

420 if $setenv; then
421     cleanup_env
422 else
423     exec ${SHELL:-sh} "$@"

```

```

424 fi

```