

new/usr/src/pkg/manifests/developer-build-onbld.mf

1

```
*****
10689 Sun Jan 26 21:59:57 2014
new/usr/src/pkg/manifests/developer-build-onbld.mf
4525 remove last vestiges of tonic
*****
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) 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright 2010, Richard Lowe
25 # Copyright 2012, Piotr Jasiukajtis
26 #
27 #
28 set name=pkg.fmri value=pkg:/developer/build/onbld@$(PKGVERS)
29 set name=pkg.description value="tools used to build the OS-Net consolidation"
30 set name=pkg.summary value="OS-Net Build Tools"
31 set name=info.classification \
32     value="org.opensolaris.category.2008:Development/Distribution Tools"
33 #
34 #
35 # This package should not be incorporated. This allows the tools
36 # to be upgraded without upgrading the entire system.
37 #
38 set name=org.opensolaris.noincorp value=true
39 set name=variant.arch value=$(ARCH)
40 dir path=opt group=sys
41 dir path=opt/onbld
42 dir path=opt/onbld/bin
43 dir path=opt/onbld/bin/$(ARCH)
44 dir path=opt/onbld/env
45 dir path=opt/onbld/etc
46 dir path=opt/onbld/etc/exception_lists
47 dir path=opt/onbld/gk
48 dir path=opt/onbld/lib
49 dir path=opt/onbld/lib/$(ARCH)
50 dir path=opt/onbld/lib/perl
51 dir path=opt/onbld/lib/python2.6
52 dir path=opt/onbld/lib/python2.6/onbld
53 dir path=opt/onbld/lib/python2.6/onbld/Checks
54 dir path=opt/onbld/lib/python2.6/onbld/Scm
55 dir path=opt/onbld/lib/python2.6/onbld/hgext
56 dir path=opt/onbld/man
57 dir path=opt/onbld/man/man1
58 $(i386_ONLY)file path=opt/onbld/bin/$(ARCH)/aw mode=0555
59 $(sparc_ONLY)file path=opt/onbld/bin/$(ARCH)/chk4ubin mode=0555
60 file path=opt/onbld/bin/$(ARCH)/codereview mode=0555
61 file path=opt/onbld/bin/$(ARCH)/cscope-fast mode=0555
```

new/usr/src/pkg/manifests/developer-build-onbld.mf

2

```
62 file path=opt/onbld/bin/$(ARCH)/ctfconvert mode=0555
63 file path=opt/onbld/bin/$(ARCH)/ctfdump mode=0555
64 file path=opt/onbld/bin/$(ARCH)/ctfmerge mode=0555
65 file path=opt/onbld/bin/$(ARCH)/ctfstabs mode=0555
66 file path=opt/onbld/bin/$(ARCH)/ctfstrip mode=0555
67 file path=opt/onbld/bin/$(ARCH)/cw mode=0555
68 $(i386_ONLY)file path=opt/onbld/bin/$(ARCH)/elfextract mode=0555
69 file path=opt/onbld/bin/$(ARCH)/findunref mode=0555
70 $(sparc_ONLY)file path=opt/onbld/bin/$(ARCH)/forth mode=0555
71 $(sparc_ONLY)file path=opt/onbld/bin/$(ARCH)/forth_preload.so.1 mode=0555
72 file path=opt/onbld/bin/$(ARCH)/install mode=0555
73 file path=opt/onbld/bin/$(ARCH)/lintdump mode=0555
74 $(i386_ONLY)file path=opt/onbld/bin/$(ARCH)/mbh_patch mode=0555
75 file path=opt/onbld/bin/$(ARCH)/ndrgen mode=0555
76 file path=opt/onbld/bin/$(ARCH)/ndrgen1 mode=0555
77 file path=opt/onbld/bin/$(ARCH)/pmodes mode=0555
78 file path=opt/onbld/bin/$(ARCH)/protocmp mode=0555
79 file path=opt/onbld/bin/$(ARCH)/protolist mode=0555
80 $(sparc_ONLY)file path=opt/onbld/bin/$(ARCH)/stabs mode=0555
81 $(sparc_ONLY)file path=opt/onbld/bin/$(ARCH)/tokenize mode=0555
82 $(sparc_ONLY)file path=opt/onbld/bin/$(ARCH)/tokenize.exe mode=0555
83 file path=opt/onbld/bin/Install mode=0555
84 file path=opt/onbld/bin/bindrop mode=0555
84 file path=opt/onbld/bin/bldenv mode=0555
85 file path=opt/onbld/bin/bringovercheck mode=0555
86 file path=opt/onbld/bin/build_cscope mode=0555
87 file path=opt/onbld/bin/cddlchk mode=0555
88 file path=opt/onbld/bin/check_rtime mode=0555
89 file path=opt/onbld/bin/checkpaths mode=0555
90 file path=opt/onbld/bin/checkproto mode=0555
91 file path=opt/onbld/bin/copyrightchk mode=0555
92 file path=opt/onbld/bin/cryptodrop mode=0555
92 file path=opt/onbld/bin/cstyle mode=0555
93 file path=opt/onbld/bin/ctfcvtptbl mode=0555
94 file path=opt/onbld/bin/ctffindmod mode=0555
95 file path=opt/onbld/bin/elfcmp mode=0555
96 file path=opt/onbld/bin/elfsigncmp mode=0555
97 file path=opt/onbld/bin/find_elf mode=0555
98 file path=opt/onbld/bin/findcrypto mode=0555
99 file path=opt/onbld/bin/flag.flp mode=0555
100 file path=opt/onbld/bin/genoffsets mode=0555
101 file path=opt/onbld/bin/get_depend_info mode=0555
102 file path=opt/onbld/bin/git-pbchk mode=0555
103 file path=opt/onbld/bin/hdrchk mode=0555
104 file path=opt/onbld/bin/hg-active mode=0555
105 file path=opt/onbld/bin/hgsetup mode=0555
106 file path=opt/onbld/bin/interface_check mode=0555
107 file path=opt/onbld/bin/interface_cmp mode=0555
108 file path=opt/onbld/bin/jstyle mode=0555
109 file path=opt/onbld/bin/make_pkg_db mode=0555
110 file path=opt/onbld/bin/mapfilechk mode=0555
113 file path=opt/onbld/bin/mkreadme_osol mode=0555
114 file path=opt/onbld/bin/mktpl mode=0555
111 file path=opt/onbld/bin/nightly mode=0555
112 file path=opt/onbld/bin/onu mode=0555
113 file path=opt/onbld/bin/protocmp.terse mode=0555
114 file path=opt/onbld/bin/sccscheck mode=0555
115 file path=opt/onbld/bin/signit mode=0555
116 file path=opt/onbld/bin/signproto mode=0555
117 file path=opt/onbld/bin/validate_flg mode=0555
118 file path=opt/onbld/bin/validate_paths mode=0555
119 file path=opt/onbld/bin/validate_pkg mode=0555
120 file path=opt/onbld/bin/wdiff mode=0555
121 file path=opt/onbld/bin/webrev mode=0555
122 file path=opt/onbld/bin/which_scm mode=0555
123 file path=opt/onbld/bin/ws mode=0555
```

```

124 file path=opt/onbld/bin/wsdiff mode=0555
125 file path=opt/onbld/bin/xref mode=0555
126 file path=opt/onbld/bin/xref.mk
127 file path=opt/onbld/env/developer
128 file path=opt/onbld/env/gatekeeper
129 file path=opt/onbld/env/illumos
130 file path=opt/onbld/etc/SampleLinks
131 file path=opt/onbld/etc/SamplePkgLinks
132 file path=opt/onbld/etc/exception_lists/check_rtime
133 file path=opt/onbld/etc/exception_lists/interface_check
134 file path=opt/onbld/etc/exception_lists/interface_cmp
135 file path=opt/onbld/etc/hgstyle
136 file path=opt/onbld/etc/its.conf
137 file path=opt/onbld/etc/its.reg
138 file path=opt/onbld/gk/.cshrc
139 file path=opt/onbld/gk/.login
140 file path=opt/onbld/gk/gen_make.machines mode=0755
141 file path=opt/onbld/lib/${ARCH}/libdwarf.so.1
142 file path=opt/onbld/lib/perl/onbld_elfmod.pm
143 file path=opt/onbld/lib/perl/onbld_elfmod_vertype.pm
144 file path=opt/onbld/lib/python2.6/onbld/Checks/CStyle.py mode=0444
145 file path=opt/onbld/lib/python2.6/onbld/Checks/CStyle.pyc mode=0444
146 file path=opt/onbld/lib/python2.6/onbld/Checks/Cddl.py mode=0444
147 file path=opt/onbld/lib/python2.6/onbld/Checks/Cddl.pyc mode=0444
148 file path=opt/onbld/lib/python2.6/onbld/Checks/CmtBlk.py mode=0444
149 file path=opt/onbld/lib/python2.6/onbld/Checks/CmtBlk.pyc mode=0444
150 file path=opt/onbld/lib/python2.6/onbld/Checks/Comments.py mode=0444
151 file path=opt/onbld/lib/python2.6/onbld/Checks/Comments.pyc mode=0444
152 file path=opt/onbld/lib/python2.6/onbld/Checks/Copyright.py mode=0444
153 file path=opt/onbld/lib/python2.6/onbld/Checks/Copyright.pyc mode=0444
154 file path=opt/onbld/lib/python2.6/onbld/Checks/DbLookups.py mode=0444
155 file path=opt/onbld/lib/python2.6/onbld/Checks/DbLookups.pyc mode=0444
156 file path=opt/onbld/lib/python2.6/onbld/Checks/HdrChk.py mode=0444
157 file path=opt/onbld/lib/python2.6/onbld/Checks/HdrChk.pyc mode=0444
158 file path=opt/onbld/lib/python2.6/onbld/Checks/JStyle.py mode=0444
159 file path=opt/onbld/lib/python2.6/onbld/Checks/JStyle.pyc mode=0444
160 file path=opt/onbld/lib/python2.6/onbld/Checks/Keywords.py mode=0444
161 file path=opt/onbld/lib/python2.6/onbld/Checks/Keywords.pyc mode=0444
162 file path=opt/onbld/lib/python2.6/onbld/Checks/Mapfile.py mode=0444
163 file path=opt/onbld/lib/python2.6/onbld/Checks/Mapfile.pyc mode=0444
164 file path=opt/onbld/lib/python2.6/onbld/Checks/ProcessCheck.py mode=0444
165 file path=opt/onbld/lib/python2.6/onbld/Checks/ProcessCheck.pyc mode=0444
166 file path=opt/onbld/lib/python2.6/onbld/Checks/__init__.py mode=0444
167 file path=opt/onbld/lib/python2.6/onbld/Checks/__init__.pyc mode=0444
168 file path=opt/onbld/lib/python2.6/onbld/Scm/Backup.py mode=0444
169 file path=opt/onbld/lib/python2.6/onbld/Scm/Backup.pyc mode=0444
170 file path=opt/onbld/lib/python2.6/onbld/Scm/Version.py mode=0444
171 file path=opt/onbld/lib/python2.6/onbld/Scm/Version.pyc mode=0444
172 file path=opt/onbld/lib/python2.6/onbld/Scm/Workspace.py mode=0444
173 file path=opt/onbld/lib/python2.6/onbld/Scm/Workspace.pyc mode=0444
174 file path=opt/onbld/lib/python2.6/onbld/Scm/__init__.py mode=0444
175 file path=opt/onbld/lib/python2.6/onbld/Scm/__init__.pyc mode=0444
176 file path=opt/onbld/lib/python2.6/onbld/__init__.py mode=0444
177 file path=opt/onbld/lib/python2.6/onbld/__init__.pyc mode=0444
178 file path=opt/onbld/lib/python2.6/onbld/hgext/__init__.py mode=0444
179 file path=opt/onbld/lib/python2.6/onbld/hgext/__init__.pyc mode=0444
180 file path=opt/onbld/lib/python2.6/onbld/hgext/cdm.py mode=0444
181 file path=opt/onbld/man/man1/Install.1
182 file path=opt/onbld/man/man1/bldenv.1
183 file path=opt/onbld/man/man1/bringovercheck.1
184 file path=opt/onbld/man/man1/cddlchk.1
185 file path=opt/onbld/man/man1/check_rtime.1
186 file path=opt/onbld/man/man1/checkpaths.1
187 file path=opt/onbld/man/man1/codereview.1
188 file path=opt/onbld/man/man1/cstyle.1
189 file path=opt/onbld/man/man1/cw.1

```

```

190 file path=opt/onbld/man/man1/find_elf.1
191 file path=opt/onbld/man/man1/findunref.1
192 file path=opt/onbld/man/man1/flg.flp.1
193 file path=opt/onbld/man/man1/get_depend_info.1
194 file path=opt/onbld/man/man1/git-pbchk.1
195 file path=opt/onbld/man/man1/hdrchk.1
196 file path=opt/onbld/man/man1/hgsetup.1
197 file path=opt/onbld/man/man1/interface_check.1
198 file path=opt/onbld/man/man1/interface_cmp.1
199 file path=opt/onbld/man/man1/jstyle.1
200 file path=opt/onbld/man/man1/lintdump.1
201 file path=opt/onbld/man/man1/make_pkg_db.1
202 file path=opt/onbld/man/man1/mapfilechk.1
203 file path=opt/onbld/man/man1/ndrgen.1
204 file path=opt/onbld/man/man1/nightly.1
205 file path=opt/onbld/man/man1/onu.1
206 file path=opt/onbld/man/man1/scscscheck.1
207 file path=opt/onbld/man/man1/signit.1
208 file path=opt/onbld/man/man1/signproto.1
209 file path=opt/onbld/man/man1/webrev.1
210 file path=opt/onbld/man/man1/which_scm.1
211 file path=opt/onbld/man/man1/ws.1
212 file path=opt/onbld/man/man1/wsdiff.1
213 file path=opt/onbld/man/man1/xref.1
214 hardlink path=opt/onbld/bin/${ARCH}/install.bin target=./install
215 legacy pkg=SUNWonbld desc="tools used to build the OS-Net consolidation" \
216   name="OS-Net Build Tools" version=11.11,REV=2009.10.22
217 license cr_Sun license=cr_Sun
218 license lic_CDDL license=lic_CDDL
219 license usr/src/tools/ctf/dwarf/THIRDPARTYLICENSE \
220   license=usr/src/tools/ctf/dwarf/THIRDPARTYLICENSE
221 license usr/src/tools/onbld/THIRDPARTYLICENSE \
222   license=usr/src/tools/onbld/THIRDPARTYLICENSE
223 link path=opt/onbld/bin/git-nits target=git-pbchk
224 link path=opt/onbld/lib/python target=python2.6
225 link path=opt/onbld/man/man1/git-nits.1 target=git-pbchk.1
226 # webrev(1) requires ps2pdf
227 depend fmri=print/filter/ghostscript type=require
228 # hgsetup(1) uses check-hostname(1) and nightly sendmail(1M)
229 depend fmri=service/network/smtp/sendmail type=require
230 # nightly(1) uses wget
231 depend fmri=web/wget type=require

```

new/usr/src/req.flg

1

1174 Sun Jan 26 21:59:58 2014

new/usr/src/req.flg

4525 remove last vestiges of tonic

```
1 #!/bin/sh
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 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
24 # Use is subject to license terms.
25 #
27 echo_file usr/src/Makefile
28 echo_file usr/src/Targetdirs
29 echo_file usr/src/Makefile.master
30 echo_file usr/src/Makefile.noget
31 echo_file usr/src/Makefile.master.64
32 echo_file usr/src/Makefile.msg.targ
33 echo_file usr/src/Makefile.psm
34 echo_file usr/src/Makefile.psm.targ
35 echo_file usr/closed/cmd/cmd-crypto/etc/certs/SUNWosnetCF
36 echo_file usr/closed/cmd/cmd-crypto/etc/certs/SUNWosnetSE
37 echo_file usr/closed/cmd/cmd-crypto/etc/keys/SUNWosnetCF
38 echo_file usr/closed/cmd/cmd-crypto/etc/keys/SUNWosnetSE
```

```
*****
23727 Sun Jan 26 21:59:58 2014
new/usr/src/tools/scripts/Install.sh
4526 nightly contains a great deal of effectively dead code
*****
_____unchanged_portion_omitted_____
```

```
374 #
375 # usage: fixcrypto listfile ctop
376 # Message entries in listfile for crypto modules, so that they point
377 # into ctop.
378 #
379 function fixcrypto {
380     typeset listfile=$1
381     typeset ctop=$2
382
383     typeset ccontents=/tmp/crypto-toc$$
384     find "$ctop" -type f -print > $ccontents
385     typeset root=root_${MACH}
386     [ "$OBJD" = obj ] && root=root_${MACH}-nd
387
388     grep -v ^MOD $listfile > $listfile.no-mod
389     grep ^MOD $listfile | while read tag srcdir module targdir size impl; do
390         #
391         # We don't just grep for ${OBJD}$size/$module because
392         # there can be generic and platform-dependent versions
393         # of a module.
394         #
395         newsrfile=$(grep -w $root/$targdir/${OBJD}$size/$module $cconte
396         if [ -n "$newsrfile" ]; then
397             # srcdir doesn't include final objNN or debugNN
398             echo $tag $module $targdir $size $impl \
399             $(dirname $(dirname "$newsrfile"))
400         else
401             echo $tag $module $targdir $size $impl $srcdir
402         fi
403     done > $listfile.mod
404     cat $listfile.mod $listfile.no-mod > $listfile
405
406     rm -f $listfile.mod
407     rm -f $listfile.no-mod
408     rm -f $ccontents
409 }
410
411 #
412 # Copy a module, or create a link, as needed.
413 #
```

```
377 function copymod {
378     case $1 in
379     MOD)
380         targdir=$INSTALL_FILES/$4
381         tstmkdir $targdir
382         target=$targdir/$3
383         verbose "$INSTALL_CP $2/${OBJD}$5/$3 $target"
384         $INSTALL_CP $2/${OBJD}$5/$3 $target || \
385             fail "can't create $target"
386         ;;
387     SYMLINK)
388         targdir=$INSTALL_FILES/$4
389         tstmkdir $targdir
390         target=$targdir/$5
391         rm -f $target
392         verbose "ln -s $3 $target"
393         ln -s $3 $target || fail "can't create $target"
394         ;;
395     *)
396         fail "unrecognized modlist entry: $*"
397         ;;
398     esac
399 }
```

```
395 LINK)
396     targdir=$INSTALL_FILES/$5
397     tstmkdir $targdir
398     target=$targdir/$6
399     rm -f $target
400     verbose "ln $INSTALL_FILES/$3/$4 $target"
401     ln $INSTALL_FILES/$3/$4 $target || fail "can't create $target"
402     ;;
403 CONF)
404     target=$INSTALL_FILES/$3
405     tstmkdir `dirname $target`
406     conffile=`basename $3`
407     verbose "$INSTALL_CP $4/$conffile $target"
408     $INSTALL_CP $4/$conffile $target
409     ;;
410 *)
411     fail "unrecognized modlist entry: $*"
412     ;;
413 esac
414 }
_____unchanged_portion_omitted_____
440 #
441 # Copy kernel modules to $INSTALL_DIR
442 #
443
444 function copy_kernel {
445
446     case $KARCH in
447     sun4*)
448         ISA=sparc; MACH=sparc ;;
449     i86*)
450         ISA=intel; MACH=i386 ;;
451     *)
452         fail "${KARCH}: invalid kernel architecture";;
453     esac
454     export MACH
455
456     if [ "$GLOM" = "no" ]; then
457         verbose "Source = $UTS, ISA = $ISA, kernel = $KARCH"
458     else
459         verbose "Source = $UTS, ISA = $ISA, kernel = $KARCH, impl = $IMP"
460     fi
461
462     test -d $KARCH || fail "${KARCH}: invalid kernel architecture"
463     test -d $ISA || fail "${ISA}: invalid instruction set architecture"
464
465     tstmkdir $INSTALL_FILES
466     rm -rf $modstatedir
467     tstmkdir $modstatedir
468     export MODSTATE=$modstatedir/state
469
470     #
471     # Figure out which "make" to use. dmake is faster than serial
472     # make, but dmake 7.3 has a bug that causes it to lose log
473     # output, which means the modlist might be incomplete.
474     #
475     make=dmake
476     dmvers=`$make -version`
477     if [ $? -ne 0 ]; then
478         make=/usr/ccs/bin/make
479     elif [[ $dmvers = *Distributed?Make?7.3* ]]; then
480         unset make
481         searchpath="/ws/onnv-tools/SUNWspro/SOS10/bin
482         /opt/SUNWspro/SOS10/bin
483         /opt/SUNWspro/bin"
484         for dmpath in $searchpath; do
485             verbose "Trying $dmpath/dmake"
486             if [ -x $dmpath/dmake ]; then
```

```

484         dmvers=`$dmpath/dmake -version`
485         if [[ $dmvers != *Distributed?Make?7.3* ]]; then
486             make="$dmpath/dmake"
487             break;
488         fi
489     fi
490 done
491 if [ -z $make ]; then
492     make=/usr/ccs/bin/make
493     echo "Warning: dmake 7.3 doesn't work with Install;" \
494         "using $make"
495 fi
496
497 #
498 # Get a list of all modules, configuration files, and links
499 # that we might want to install.
500 #
501 verbose "Building module list..."
502 (cd $KARCH; MAKEFLAGS=e $make -K $MODSTATE modlist.karch) | \
503     egrep "^MOD|^CONF|^LINK|^SYMLINK" > $modlist
504 [ "$VERBOSE" = "V" ] && cat $modlist
505 check_modlist $modlist
506 if [ -n "$ON_CRYPTO_BINS" ]; then
507     cryptotar="$ON_CRYPTO_BINS"
508     if [ "$OBJD" = obj ]; then
509         isa=$(uname -p)
510         cryptotar=$(echo "$ON_CRYPTO_BINS" |
511             sed -e s/.$isa.tar.bz2/-nd.$isa.tar.bz2/)
512     fi
513     [ -f "$cryptotar" ] || fail "crypto ($cryptotar) doesn't exist"
514     cryptotree=$(mktemp -d /tmp/crypto.XXXXXX)
515     [ -n "$cryptotree" ] || fail "can't create tree for crypto"
516     unpack_crypto "$cryptotar" "$cryptotree"
517     #
518     # fixcrypto must come before fixglom, because
519     # fixcrypto uses the unglommed path to find things in
520     # the unpacked crypto.
521     #
522     fixcrypto $modlist "$cryptotree"
523 fi
524 if [ "$GLOM" = "yes" ]; then
525     fixglom $modlist $GLOMNAME
526     filtimpl $modlist $IMPL
527 fi
528 if [[ -n "$files" && "$files" != All ]]; then
529     filtmod $modlist "$files"
530 fi
531
532 #
533 # Copy modules and create links. For architectures with both
534 # 32- and 64-bit modules, we'll likely have duplicate
535 # configuration files, so do those after filtering out the
536 # duplicates.
537 #
538 verbose "Copying files to ${INSTALL_FILES}..."
539
540 #
541 # The IFS is reset to the newline character so we can buffer the
542 # output of grep without piping it directly to copymod, otherwise
543 # if fail() is called, then it will deadlock in fail()'s wait call
544 #
545 OIFS="$IFS"
546 IFS=""
547 "
548 set -- `grep -v "^CONF" $modlist`;

```

```

532     IFS="$OIFS"
533     for onemod in "$@"; do
534         copymod $onemod
535     done
536
537 OIFS="$IFS"
538 IFS=""
539 "
540 set -- `grep "^CONF" $modlist | sort | uniq`;
541 IFS="$OIFS"
542 for onemod in "$@"; do
543     copymod $onemod
544 done
545
546 #
547 # Add the glommed kernel name to the root archive
548 #
549 if [[ $GLOM == "yes" ]];
550 then
551     filelist="$INSTALL_FILES/etc/boot/solaris/filelist.ramdisk"
552     mkdir -p `dirname $filelist`
553     echo "platform/$KARCH/$GLOMNAME" >$filelist
554 fi
555
556 STATE=1 # all kernel modules copied correctly
557 save_state
558 }
559
560 unchanged_portion_omitted
561
562 function copy_kmdb {
563     typeset kmdbtgtmdir=$INSTALL_FILES/platform/$KARCH/$GLOMNAME/misc
564     typeset bitdirs=
565     typeset isadir=
566     typeset b64srcdir=
567     typeset b64tgtmdir=
568     typeset b32srcdir=
569     typeset b32tgtmdir=
570     typeset machdir=
571     typeset platdir=
572
573     if [[ $KMDB = "no" || ! -d $SRC/cmd/mdb ]]; then
574         # The kmdb copy was suppressed or the workspace doesn't contain
575         # the mdb subtree. Either way, there's nothing to do.
576         STATE=2
577         save_state
578         return
579     fi
580
581     if [[ $(mach) = "i386" ]]; then
582         isadir="intel"
583         b64srcdir="amd64"
584         b64tgtmdir="amd64"
585         b32srcdir="ia32"
586         b32tgtmdir="."
587     else
588         isadir="sparc"
589         b64srcdir="v9"
590         b64tgtmdir="sparcv9"
591         b32srcdir="v7"
592         b32tgtmdir="."
593     fi
594
595     typeset foundkmdb=no
596     typeset kmdbpath=
597     typeset destdir=

```

```

671 platdir=$INSTALL_FILES/platform/$KARCH/$GLOMNAME
672 if [[ $GLOM = "yes" ]] ; then
673     machdir=$platdir
674 else
675     machdir=$INSTALL_FILES/kernel
676 fi
677
678 srctrees=$SRC
679 if [ -z "$ON_CRYPTO_BINS" ] ; then
680     echo "Warning: ON_CRYPTO_BINS not set; pre-signed" \
681         "crypto not provided."
682 fi
683 if [[ $WANT64 = "yes" ]] ; then
684     # kmdbmod for sparc and x86 are built and installed
685     # in different places
686     if [[ $(mach) = "i386" ]] ; then
687         kmdbpath=$SRC/cmd/mdb/$isadir/$b64srcdir/kmdb/kmdbmod
688         destdir=$machdir/misc/$b64tgtmdir
689     else
690         kmdbpath=$SRC/cmd/mdb/$KARCH/$b64srcdir/kmdb/kmdbmod
691         destdir=$platdir/misc/$b64tgtmdir
692     fi
693
694     if kmdb_copy_kmdbmod $kmdbpath $destdir ; then
695         foundkmdb="yes"
696
697         for tree in $srctrees; do
698             kmdb_copy_machkmods \
699                 $tree/cmd/mdb/$isadir/$b64srcdir \
700                 $machdir/kmdb/$b64tgtmdir
701             kmdb_copy_karchkmods $tree/cmd/mdb/$KARCH \
702                 $platdir/kmdb/$b64tgtmdir $b64srcdir
703         done
704     fi
705
706     if [[ $WANT32 = "yes" ]] ; then
707         kmdbpath=$SRC/cmd/mdb/$isadir/$b32srcdir/kmdb/kmdbmod
708         destdir=$machdir/misc/$b32tgtmdir
709
710         if kmdb_copy_kmdbmod $kmdbpath $destdir ; then
711             foundkmdb="yes"
712
713             for tree in $srctrees; do
714                 kmdb_copy_machkmods \
715                     $tree/cmd/mdb/$isadir/$b32srcdir \
716                     $machdir/kmdb/$b32tgtmdir
717                 kmdb_copy_karchkmods $tree/cmd/mdb/$KARCH \
718                     $platdir/kmdb/$b32tgtmdir $b32srcdir
719             done
720         fi
721
722         # A kmdb-less workspace isn't fatal, but it is potentially problematic,
723         # as the changes made to uts may have altered something upon which kmdb
724         # depends. We will therefore remind the user that they haven't built it
725         # yet.
726         if [[ $foundkmdb != "yes" ]] ; then
727             echo "WARNING: kmdb isn't built, and won't be included"
728         fi
729
730     STATE=2
731     save_state
732     return
733 }

```

unchanged portion omitted

```

*****
3584 Sun Jan 26 21:59:59 2014
new/usr/src/tools/scripts/Makefile
4525 remove last vestiges of tonic
*****
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 # Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
23 #
24 # Copyright 2010, Richard Lowe

26 SHELL=/usr/bin/ksh93

28 SHFILES= \
29     Install \
30     bindrop \
31     bldenv \
32     build_cscope \
33     bringovercheck \
34     checkpaths \
35     checkproto \
36     cryptodrop \
37     cstyle \
38     elfcmp \
39     flg.flp \
40     genoffsets \
41     hgsetup \
42     nightly \
43     onu \
44     protocmp terse \
45     sccscheck \
46     webrev \
47     which_scm \
48     ws \
49     xref

49 PERLFILES= \
50     check_runtime \
51     find_elf \
52     interface_check \
53     interface_cmp \
54     jstyle \
55     mkreadme_osol \
56     mktpl \
57     validate_flg \
58     validate_paths \
59     wdiff

```

```

59 PERLMODULES= \
60     onbld_elfmod.pm \
61     onbld_elfmod_vertype.pm

64 PYFILES= \
65     cddlchk \
66     copyrightchk \
67     git-pbchk \
68     hdrchk \
69     hg-active \
70     mapfilechk \
71     validate_pkg \
72     wsdiff

74 SCRIPTLINKS= \
75     git-nits

77 MANIFILES= \
78     Install.1 \
79     bldenv.1 \
80     bringovercheck.1 \
81     cddlchk.1 \
82     checkpaths.1 \
83     check_runtime.1 \
84     cstyle.1 \
85     find_elf.1 \
86     flg.flp.1 \
87     git-pbchk.1 \
88     hdrchk.1 \
89     interface_check.1 \
90     interface_cmp.1 \
91     hgsetup.1 \
92     jstyle.1 \
93     mapfilechk.1 \
94     nightly.1 \
95     onu.1 \
96     sccscheck.1 \
97     webrev.1 \
98     which_scm.1 \
99     ws.1 \
100    wsdiff.1 \
101    xref.1

103 MANLINKS= \
104     git-nits.1

106 MAKEFILES= \
107     xref.mk

109 ETCFILES= \
110     hgstyle \
111     its.conf \
112     its.reg

114 EXCEPTFILES= \
115     check_runtime \
116     interface_check \
117     interface_cmp

119 CLEANFILES= $(SHFILES) $(PERLFILES) $(PYFILES) bldenv.1

121 include ../Makefile.tools

123 ROOTONBLDSRIPTLINKS = $(SCRIPTLINKS:%=$(ROOTONBLDBIN)/%)

```

```
124 ROOTONBLDMAN1LINKS = $(MAN1LINKS:%=$(ROOTONBLDMAN1)/%)

126 $(ROOTONBLDETCFILES) := FILEMODE= 644
127 $(ROOTONBLDEXCEPTFILES) := FILEMODE= 644
128 $(ROOTONBLDPERLMODULES) := FILEMODE= 644
129 $(ROOTONBLDMAKEFILES) := FILEMODE= 644
130 $(ROOTONBLDMAN1FILES) := FILEMODE= 644

132 .KEEP_STATE:

134 all: $(SHFILES) $(PERLFILES) $(PERLMODULES) $(PYFILES) \
135      $(MAN1FILES) $(MAKEFILES)

137 $(ROOTONBLDBIN)/git-nits:
138     $(RM) $(ROOTONBLDBIN)/git-nits
139     $(SYMLINK) git-pbchk $(ROOTONBLDBIN)/git-nits

141 $(ROOTONBLDMAN1)/git-nits.1:
142     $(RM) $(ROOTONBLDMAN1)/git-nits.1
143     $(SYMLINK) git-pbchk.1 $(ROOTONBLDMAN1)/git-nits.1

145 install: all .WAIT $(ROOTONBLDSSHFILES) $(ROOTONBLDPERLFILES) \
146            $(ROOTONBLDPERLMODULES) $(ROOTONBLDPYFILES) \
147            $(ROOTONBLDSCRIPTLINKS) $(ROOTONBLDMAN1FILES) \
148            $(ROOTONBLDMAKEFILES) $(ROOTONBLDETCFILES) \
149            $(ROOTONBLDEXCEPTFILES) $(ROOTONBLDMAN1LINKS)

151 clean:
152     $(RM) $(CLEANFILES)

154 bldenv: bldenv.sh stdenv.sh
155     $(RM) "$@"
156     sed -e '/# STDENV_START/ r stdenv.sh' bldenv.sh > "$@"
157     # Check for shell lint and fail if we hit warnings
158     shlintout=$$( /usr/bin/ksh93 -n "$@" 2>&1 ) ; \
159     [[ "${shlintout}" != "" ]] && \
160     { print -r -- "${shlintout}" ; false ; } || true
161     $(CHMOD) +x "$@"

163 bldenv.1: bldenv
164     $(RM) "$@"
165     (set +o errexit ; ksh93 $? --nroff ; true) 2>&1 | \
166     sed 's/\.DS/.nf/g;s/\.DE/.fi/' > "$@"

168 nightly: nightly.sh stdenv.sh
169     $(RM) "$@"
170     sed -e '/# STDENV_START/ r stdenv.sh' nightly.sh > nightly
171     $(CHMOD) +x "$@"

173 include ../Makefile.targ
```


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

1

```
*****
10066 Sun Jan 26 21:59:59 2014
new/usr/src/tools/scripts/bldenv.sh
4526 nightly contains a great deal of effectively dead code
*****
```

unchanged portion omitted

```
109 [+SEE ALSO?\bnightly\b(1)]
110 '

112 # main
113 builtin basename

115 # boolean flags (true/false)
116 typeset flags=(
117     typeset c=false
118     typeset f=false
119     typeset d=false
120     typeset O=false
121     typeset o=false
122     typeset t=true
123     typeset s=(
124         typeset e=false
125         typeset h=false
126         typeset d=false
127         typeset o=false
128     )
129 )

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

133 OPTIND=1
134 SUFFIX="-nd"

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

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

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

165 # clear environment variables we know to be bad for the build
166 unset \
167     CH \
168 #endif /* ! codereview */
```

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

2

```
169     LD_OPTIONS \
170     LD_LIBRARY_PATH \
171     LD_AUDIT \
172     LD_BIND_NOW \
173     LD_BREADTH \
174     LD_CONFIG \
175     LD_DEBUG \
176     LD_FLAGS \
177     LD_LIBRARY_PATH_64 \
178     LD_NOVERSION \
179     LD_ORIGIN \
180     LD_LOADFLTR \
181     LD_NOAUXFLTR \
182     LD_NOCONFIG \
183     LD_NODIRCONFIG \
184     LD_NOOBJALTER \
185     LD_PRELOAD \
186     LD_PROFILE \
187     CONFIG \
188     GROUP \
189     OWNER \
190     REMOTE \
191     ENV \
192     ARCH \
193     CLASSPATH

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

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

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

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

229 # must match the getopts in nightly.sh
230 OPTIND=1
231 NIGHTLY_OPTIONS="-${NIGHTLY_OPTIONS#-}"
232 while getopts '+0ABCDdFfGIilMmNnpRrtUuw' FLAG "$NIGHTLY_OPTIONS"
167 while getopts '+0AaBCDDdFfGIilMmNnopRrtUuWwXxz' FLAG "$NIGHTLY_OPTIONS"
233 do
```

```

234     case "$FLAG" in
170         o)    flags.o=true ;;
171         +o)   flags.o=false ;;
235         t)    flags.t=true ;;
236         +t)   flags.t=false ;;
237         *)    ;;
238     esac
239 done

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

251 print 'Build type is \c'
252 if ${flags.d} ; then
253     print 'DEBUG'
254     unset RELEASE_BUILD
255     unset EXTRA_OPTIONS
256     unset EXTRA_CFLAGS
257 else
258     # default is a non-DEBUG build
259     print 'non-DEBUG'
260     export RELEASE_BUILD=
261     unset EXTRA_OPTIONS
262     unset EXTRA_CFLAGS
263 fi

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

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

274 if [[ -n "${MAKE}" ]]; then
275     if [[ -x "${MAKE}" ]]; then
276         export PATH="$(dirname -- "${MAKE}"):$PATH"
277     else
278         print "\$MAKE (${MAKE}) is not a valid executable"
279         exit 1
280     fi
281 fi

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

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

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

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

296     export CTFCVPTBL="${TOOLS_PROTO}/opt/onbld/bin/ctfcvptbl"
297     export CTFINDMOD="${TOOLS_PROTO}/opt/onbld/bin/ctfindmod"

```

```

299     PATH="${TOOLS_PROTO}/opt/onbld/bin/${MACH}:${PATH}"
300     PATH="${TOOLS_PROTO}/opt/onbld/bin:${PATH}"
301     export PATH
302 fi

304 export DMAKE_MODE=${DMAKE_MODE:-parallel}

243 if "${flags.o}" ; then
244     export CH=
245 else
246     unset CH
247 fi
306 DEF_STRIPFLAG="-s"

308 TMPDIR="/tmp"

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

310 export \
311     PATH TMPDIR \
312     POUND_SIGN \
313     DEF_STRIPFLAG \
314     RELEASE_DATE
315 unset \
316     CFLAGS \
317     LD_LIBRARY_PATH

319 # a la ws
320 ENVLDLIBS1=
321 ENVLDLIBS2=
322 ENVLDLIBS3=
323 ENVCPPFLAGS1=
324 ENVCPPFLAGS2=
325 ENVCPPFLAGS3=
326 ENVCPPFLAGS4=
327 PARENT_ROOT=
328 PARENT_TOOLS_ROOT=

330 if [[ "${MULTI_PROTO}" != "yes" && "${MULTI_PROTO}" != "no" ]]; then
331     printf \
332         'WARNING: invalid value for MULTI_PROTO (%s); setting to "no".\n' \
333         "${MULTI_PROTO}"
334     export MULTI_PROTO="no"
335 fi

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

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

343 export \
344     ENVLDLIBS1 \
345     ENVLDLIBS2 \
346     ENVLDLIBS3 \
347     ENVCPPFLAGS1 \
348     ENVCPPFLAGS2 \
349     ENVCPPFLAGS3 \
350     ENVCPPFLAGS4 \
351     MAKEFLAGS \
352     PARENT_ROOT \
353     PARENT_TOOLS_ROOT

```

```
355 printf 'RELEASE      is %s\n' "$RELEASE"
356 printf 'VERSION      is %s\n' "$VERSION"
357 printf 'RELEASE_DATE is %s\n\n' "$RELEASE_DATE"

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

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

370 #
371 # place ourselves in a new task, respecting BUILD_PROJECT if set.
372 #
373 /usr/bin/newtask -c $$ ${BUILD_PROJECT:+-p$BUILD_PROJECT}

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

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

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

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

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

395 # not reached
```

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

1

```
*****
3633 Sun Jan 26 22:00:00 2014
new/usr/src/tools/scripts/checkpaths.sh
4525 remove last vestiges of tonic
*****
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 for ROOT in $rootlist
63 do
64     case "$ROOT" in
65         *sparc|*sparc-nd)
66             arch=sparc
67             ;;
68         *i386|*i386-nd)
69             arch=i386
70             ;;
71         *)
72             echo "$ROOT has unknown architecture." >&2
73             exit 1
74             ;;
75     esac
76     if [ -d $ROOT ]; then
77         #
78         # This is the old-style packaging exception list, from
79         # the svr4-specific usr/src/pkgdefs
80         #
81         [ -f $SRC/pkgdefs/etc/exception_list_$arch ] && \
82             validate_paths '-s/\s*' $arch'$/ ' \
83                 -e ^usr/include/ike/ -b $ROOT \
84                 $args $SRC/pkgdefs/etc/exception_list_$arch
85         #
86         # These are the new-style packaging exception lists,
87         # from the repository-wide exception_lists/ directory.
88         #
89         e="$CODEMGR_WS/exception_lists/packaging"
90         for f in $e; do
91             if [ -f $f ]; then
92                 nawk 'NF == 1 || /[ ]\+' $arch'$/ { print; }'
93                 < $f | validate_paths -b $ROOT -n $f
94             fi
95         done
96     fi
97 done
98 #
99 # Two entries in the findunref exception_list deal with things created
100 # by nightly. Otherwise, this test could be run on an unmodified (and
101 # unbuilt) workspace. We handle this by flagging the one that is
102 # present only on a built workspace (./*.out) and the one that's
103 # present only after a run of findunref (./*.ref) with ISUSED, and
104 # disabling all checks of them. The assumption is that the entries
105 # marked with ISUSED are always known to be good, thus the Latin quote
106 # at the top of the file.
107 #
108 # The exception_list is generated from whichever input files are appropriate
109 # for this workspace, so checking it obviates the need to check the inputs.
110 #
111 if [ -r $SRC/tools/findunref/exception_list ]; then
112     validate_paths -k ISUSED -r -e '^*' $SRC/tools/findunref/exception_list
113 fi
114 #
115 if [ -f $SRC/tools/opensolaris/license-list ]; then
116     sed -e 's/./.descrip/' < $SRC/tools/opensolaris/license-list | \
117         validate_paths -n SRC/tools/opensolaris/license-list
118     validate_paths -n SRC/tools/opensolaris/license-list \
119         -e ^usr/closed
120 fi
121 #
122 validate_flg -f
123 # Finally, make sure the that (req/inc).flg files are in good shape.
124 # If SCCS files are not expected to be present, though, then don't
125 # check them.
126 if [ ! -d "$CODEMGR_WS/Codemgr_wsdata" ]; then
127     f_flg='-f'
```

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

3

```
126 fi
```

```
128 validate_flg $f_flg -e ^usr/closed/
```

```
122 exit 0
```

new/usr/src/tools/scripts/nightly.1

1

```
*****
14823 Sun Jan 26 22:00:01 2014
new/usr/src/tools/scripts/nightly.1
4526 nightly contains a great deal of effectively dead code
*****
1 .\" "
2 .\" " The contents of this file are subject to the terms of the
3 .\" " Common Development and Distribution License (the "License").
4 .\" " You may not use this file except in compliance with the License.
5 .\" "
6 .\" " You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
7 .\" " or http://www.opensolaris.org/os/licensing.
8 .\" " See the License for the specific language governing permissions
9 .\" " and limitations under the License.
10 .\" "
11 .\" " When distributing Covered Code, include this CDDL HEADER in each
12 .\" " file and include the License file at usr/src/OPENSOLARIS.LICENSE.
13 .\" " If applicable, add the following below this CDDL HEADER, with the
14 .\" " fields enclosed by brackets "[]" replaced with your own identifying
15 .\" " information: Portions Copyright [yyyy] [name of copyright owner]
16 .\" "
17 .\" " CDDL HEADER END
18 .\" "
19 .\" " Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved
20 .\" " Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
21 .\" "
22 .TH nightly 1 "6 July 2010"
23 .SH NAME
24 .I nightly
25 \- build an OS-Net consolidation overnight
26 .SH SYNOPSIS
27 \fBnightly [-in] [-V VERS] <env_file>\fP
28 .LP
29 .SH DESCRIPTION
30 .IX "OS-Net build tools" "nightly" "" "\fBnightly\fP"
31 .LP
32 .I nightly,
33 the mother of all build scripts,
34 can bringover, build, archive, package, error check, and
35 generally do everything it takes to
36 turn OS/Net consolidation source code into useful stuff.
37 It is customizable to permit you to run anything from a
38 simple build to all of the cross-checking a gatekeeper
39 needs. The advantage to using
40 .I nightly
41 is that you build things correctly, consistently and
42 automatically, with the best practices; building with
43 .I nightly
44 can mean never having to say you're sorry to your
45 gatekeeper.
46 .LP
47 More
48 specifically,
49 .I nightly
50 performs the following tasks, in order, if
51 all these things are desired:
52 .LP
53 .RS
54 .TP
55 \(\bu
56 perform a "make clobber" to clean up old binaries
57 .TP
58 \(\bu
59 bringover from the identified parent gate/clone
60 .TP
61 \(\bu
```

new/usr/src/tools/scripts/nightly.1

2

```
62 perform non-DEBUG and DEBUG builds
63 .TP
64 \(\bu
65 list proto area files and compare with previous list
66 .TP
67 \(\bu
68 copy updated proto area to parent
69 .TP
70 \(\bu
71 list shared lib interface and compare with previous list
72 .TP
73 \(\bu
74 perform a "make lint" of the kernel and report errors
75 .TP
76 \(\bu
77 perform a "make check" to report hdrchk/cstyle errors
78 .TP
79 \(\bu
80 report the presence of any core files
81 .TP
82 \(\bu
83 check the ELF runtime attributes of all dynamic objects
84 .TP
85 \(\bu
86 check for unreferenced files
87 .TP
88 \(\bu
89 report on which proto area objects have changed (since the last build)
90 .TP
91 \(\bu
92 report the total build time
93 .TP
94 \(\bu
95 save a detailed log file for reference
96 .TP
97 \(\bu
98 mail the user a summary of the completed build
99 .RE
100 .LP
101 The actions of the script are almost completely determined by
102 the environment variables in the
103 .I env
104 file, the only necessary argument. This only thing you really
105 need to use
106 .I nightly
107 is an
108 .I env
109 file that does what you want.
110 .LP
111 Like most of the other build tools in usr/src/tools, this script tends
112 to change on a fairly regular basis; do not expect to be able to build
113 OS/Net with a version of nightly significantly older than your source
114 tree. It has what is effectively a Consolidation Private relationship
115 to other build tools and with many parts of the OS/Net makefiles,
116 although it may also be used to build other consolidations.
117 .LP
118 .SH NIGHTLY_OPTIONS
119 The environment variable NIGHTLY_OPTIONS controls the actions
120 .I nightly
121 will take as it proceeds.
122 The -i, -n, +t and -V options may also be used from the command
123 line to control the actions without editing your environment file.
124 The -i and -n options complete the build more quickly by bypassing
125 some actions. If NIGHTLY_OPTIONS is not set, then "-Bmt" build
126 options will be used.
```

```

128 .B Basic action options
129 .TP 10
130 .B \-D
131 Do a build with DEBUG on (non-DEBUG is built by default)
132 .TP
133 .B \-F
134 Do _not_ do a non-DEBUG build (use with -D to get just a DEBUG build)
135 .TP
136 .B \-M
137 Do not run pmodes (safe file permission checker)
138 .TP
139 .B \-i
140 Do an incremental build, suppressing the "make clobber" that by
141 default removes all existing binaries and derived files. From the
142 command line, -i also suppresses the lint pass and the cstyle/hdrchk
143 pass
144 .TP
145 .B \-n
146 Suppress the bringover so that the build will start immediately with
147 current source code
148 .TP
149 .B \-o
150 Do an "old style" (pre-S10) build using root privileges to set OWNER
151 and GROUP from the Makefiles.
152 .TP
153 .B \-p
154 Create packages for regular install
155 .TP
156 .B \-U
157 Update proto area in the parent workspace
158 .TP
159 .B \-u
160 Update the parent workspace with files generated by the build, as follows.
161 .RS
162 .TP
163 \(\bu
164 Copy proto_list_${MACH} and friends to usr/src in the parent.
165 .TP
166 \(\bu
167 When used with -f, build a usr/src/unrefmaster.out in
168 the parent by merging all the usr/src/unref-${MACH}.out files in the
169 parent.
170 .TP
171 \(\bu
172 When used with -A or -r, copy the contents of the resulting
173 ELF-data.${MACH} directory to usr/src/ELF-data.${MACH} in the parent
174 workspace.
175 .RE
176 .TP
177 .B \-m
178 Send mail to $MAILTO at end of build
179 .TP
180 .B \-t
181 Build and use the tools in $SRC/tools (default setting).
182 .TP
183 .B \+t
184 Use the build tools in "$ONBLD_TOOLS/bin".

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

```

```

190 .B \-C
191 Check for cstyle/hdrchk errors
192 .TP
193 .B \-f
194 Check for unreferenced files. Since the full workspace must be built
195 in order to accurately identify unreferenced files, -f is ignored for
196 incremental (-i) builds, or builds that do not include -l, and -p.
197 .TP
198 .B \-r
199 Check the ELF runtime attributes of all dynamic objects
200 .TP
201 .B \-l
202 Do "make lint" in $LINTDIRS (default: $SRC n)
203 .TP
204 .B \-N
205 Do not run protocmp or checkpaths (note: this option is not
206 recommended, especially in conjunction with the \-p option)
207 .TP
208 .B \-W
209 Do not report warnings (for freeware gate ONLY)
210 .TP
211 .B \-w
212 Report which proto area objects differ between this and the last build.
213 See wsdiff(1) for details. Note that the proto areas used for comparison
214 are the last ones constructed as part of the build. As an example, if both
215 a non-debug and debug build are performed (in that order), then the debug
216 proto area will be used for comparison (which might not be what you want).
217 .LP
218 .B Groups of options
219 .TP 10
220 .B \-G
221 Gate keeper default group of options (-u)
222 .TP
223 .B \-I
224 Integration engineer default group of options (-mpu)
225 .TP
226 .B \-R
227 Default group of options for building a release (-mp)

226 .LP
227 .B Miscellaneous options
228 .TP 10
229 .B \-V VERS
230 set the build version string to VERS, overriding VERSION
231 .TP
232 .B \-X
233 Copies the proto area and packages from the IHV and IHV-bin gates into the
234 nightly proto and package areas. This is only available on i386. See
235 B REALMODE ENVIRONMENT VARIABLES
236 and
237 B BUILDING THE IHV WORKSPACE
238 below.

232 .LP
233 .SH ENVIRONMENT VARIABLES
234 .LP
235 Here is a list of prominent environment variables that
236 .I nightly
237 references and the meaning of each variable.
238 .LP
239 .RE
240 .B CODEMGR_WS
241 .RS 5
242 The root of your workspace, including whatever metadata is kept by
243 the source code management system. This is the workspace in which the
244 build will be done.

```

```

245 .RE
246 .LP
247 .B PARENT_WS
248 .RS 5
249 The root of the workspace that is the parent of the
250 one being built. This is particularly relevant for configurations
251 with a main
252 workspace and build workspaces underneath it; see the
253 \-u and \-U
254 options as well as the PKGARCHIVE environment variable, for more
255 information.
256 .RE
257 .LP
258 .B BRINGOVER_WS
259 .RS 5
260 This is the workspace from which
261 .I nightly
262 will fetch sources to either populate or update your workspace;
263 it defaults to $CLONE_WS.
264 .RE
265 .LP
266 .B CLOSED_BRINGOVER_WS
267 .RS 5
268 A full Mercurial workspace has two repositories: one for open source
269 and one for closed source. If this variable is non-null,
270 .I nightly
271 will pull from the repository that it names to get the closed source.
272 It defaults to $CLOSED_CLONE_WS.
273 .RE
274 .LP
275 .B CLONE_WS
276 .RS 5
277 This is the workspace from which
278 .I nightly
279 will fetch sources by default. This is
280 often distinct from the parent, particularly if the parent is a gate.
281 .RE
282 .LP
283 .B CLOSED_CLONE_WS
284 .RS 5
285 This is the default closed-source Mercurial repository that
286 .I nightly
287 might pull from (see
288 .B CLOSED_BRINGOVER_WS
289 for details).
290 .RE
291 .LP
292 .B SRC
293 .RS 5
294 Root of OS-Net source code, referenced by the Makefiles. It is
295 the starting point of build activity. It should be expressed
296 in terms of $CODEMGR_WS.
297 .RE
298 .LP
299 .B ROOT
300 .RS 5
301 Root of the proto area for the build. The makefiles direct
302 installation of build products to this area and
303 direct references to these files by builds of commands and other
304 targets. It should be expressed in terms of $CODEMGR_WS.
305 .RE
306 .LP

```

```

288 If $MULTI_PROTO is "no", $ROOT may contain a DEBUG or non-DEBUG
289 build. If $MULTI_PROTO is "yes", $ROOT contains the DEBUG build and
290 $ROOT-nd contains the non-DEBUG build.
291 .RE
292 .LP
293 .B TOOLS_ROOT
294 .RS 5
295 Root of the tools proto area for the build. The makefiles direct
296 installation of tools build products to this area. Unless \fb+t\fr
297 is part of $NIGHTLY_OPTIONS, these tools will be used during the
298 build.
299 .LP
300 As built by nightly, this will always contain non-DEBUG objects.
301 Therefore, this will always have a -nd suffix, regardless of
302 $MULTI_PROTO.
303 .RE
304 .LP
305 .B MACH
306 .RS 5
307 The instruction set architecture of the build machine as given
308 by \fiuname -p\fp, e.g. sparc, i386.
309 .RE
310 .LP
311 .B LOCKNAME
312 .RS 5
313 The name of the file used to lock out multiple runs of
314 .I nightly .
315 This should generally be left to the default setting.
316 .RE
317 .LP
318 .B ATLOG
319 .RS 5
320 The location of the log directory maintained by
321 .I nightly .
322 This should generally be left to the default setting.
323 .RE
324 .LP
325 .B LOGFILE
326 .RS 5
327 The name of the log file in the $ATLOG directory maintained by
328 .I nightly .
329 This should generally be left to the default setting.
330 .RE
331 .LP
332 .B STAFFER
333 .RS 5
334 The non-root account to use on the build machine for the
335 bringover from the clone or parent workspace.
336 This may not be the same identify used by the SCM.
337 .RE
338 .LP
339 .B MAILTO
340 .RS 5
341 The address to be used to send completion e-mail at the end of
342 the build (for the \-m option).
343 .RE
344 .LP
345 .B MAILFROM
346 .RS 5
347 The address to be used for From: in the completion e-mail at the
348 end of the build (for the \-m option).
349 .RE
350 .LP
351 .B REF_PROTO_LIST
352 .RS 5
353 Name of file used with protocmp to compare proto area contents.

```



```

354 .RE
355 .LP
356 .B PARENT_ROOT
357 .RS 5
358 The parent root, which is the destination for copying the proto
359 area(s) when using the \-U option.
360 .RE
361 .LP
362 .B PARENT_TOOLS_ROOT
363 .RS 5
364 The parent tools root, which is the destination for copying the tools
365 proto area when using the \-U option.
366 .RE
367 .LP
368 .B RELEASE
369 .RS 5
370 The release version number to be used; e.g., 5.10.1 (Note: this is set
371 in Makefile.master and should not normally be overridden).
372 .RE
373 .LP
374 .B VERSION
375 .RS 5
376 The version text string to be used; e.g., "onnv:'date '+%Y-%m-%d'".
377 .RE
378 .LP
379 .B RELEASE_DATE
380 .RS 5
381 The release date text to be used; e.g., October 2009. If not set in
382 your environment file, then this text defaults to the output from
383 $(LC_ALL=C date +"%B %Y"); e.g., "October 2009".
384 .RE
385 .LP
386 .B RELEASE_BUILD
387 .RS 5
388 Define this to build a release with a non-DEBUG kernel.
389 Generally, let
390 .I nightly
391 set this for you based on its options.
392 .RE
393 .LP
394 .B PKGARCHIVE
395 .RS 5
396 The destination for packages. This may be relative to
397 $CODEMGR_WS for private packages or relative to $PARENT_WS
398 if you have different workspaces for different architectures
399 but want one hierarchy of packages.
400 .RE
401 .LP
402 .B MAKEFLAGS
403 .RS 5
404 Set default flags to make; e.g., -k to build all targets regardless of errors.
405 .RE
406 .LP
407 .B UT_NO_USAGE_TRACKING
408 .RS 5
409 Disables usage reporting by listed Devpro tools. Otherwise it sends mail
410 to some Devpro machine every time the tools are used.
411 .RE
412 .LP
413 .B LINTDIRS
414 .RS 5
415 Directories to lint with the \-l option.
416 .RE
417 .LP
418 .B BUILD_TOOLS
419 .RS 5

```

```

420 BUILD_TOOLS is the root of all tools including the compilers; e.g.,
421 /ws/onnv-tools. It is used by the makefile system, but not nightly.
422 .RE
423 .LP
424 .B ONBLD_TOOLS
425 .RS 5
426 ONBLD_TOOLS is the root of all the tools that are part of SUNWonbld; e.g.,
427 /ws/onnv-tools/onbld. By default, it is derived from
428 .BR BUILD_TOOLS .
429 It is used by the makefile system, but not nightly.
430 .RE
431 .LP
432 .B SPRO_ROOT
433 .RS 5
434 The gate-defined default location for the Sun compilers, e.g.
435 /ws/onnv-tools/SUNWspro. By default, it is derived from
436 .BR BUILD_TOOLS .
437 It is used by the makefile system, but not nightly.
438 .RE
439 .LP
440 .B JAVA_ROOT
441 .RS 5
442 The location for the java compilers for the build, generally /usr/java.
443 .RE
444 .LP
445 .B OPTHOME
446 .RS 5
447 The gate-defined default location of things formerly in /opt; e.g.,
448 /ws/onnv-tools. This is used by nightly, but not the makefiles.
449 .RE
450 .LP
451 .B TEAMWARE
452 .RS 5
453 The gate-defined default location for the Teamware tools; e.g.,
454 /ws/onnv-tools/SUNWspro. By default, it is derived from
455 .BR OPTHOME .
456 This is used by nightly, but not the makefiles. There is no
457 corresponding variable for Mercurial or Subversion, which are assumed
458 to be installed in the default path.
459 .RE
460 .LP
461 .B ON_CLOSED_BINS
462 .RS 5
463 OpenSolaris builds do not contain the closed source tree. Instead,
464 the developer downloads a closed binaries tree and unpacks it.
465 .B ON_CLOSED_BINS
466 tells nightly
467 where to find these closed binaries, so that it can add them into the
468 build.
469 .RE
470 .LP
471 .B ON_CRYPTO_BINS
472 .RS 5
473 This is the path to a compressed tarball that contains debug
474 cryptographic binaries that have been signed to allow execution
475 outside of Sun, e.g., $PARENT_WS/packages/$MACH/on-crypto.$MACH.bz2.
476 .I nightly
477 will automatically adjust the path for non-debug builds. This tarball
478 is needed if the closed-source tree is not present. Also, it is
479 usually needed when generating OpenSolaris deliverables from a project
480 workspace. This is because most projects do not have access to the
481 necessary key and certificate that would let them sign their own
482 cryptographic binaries.
470 .LP
471 .RE
472 .B CHECK_PATHS

```

```

472 .RS 5
473 Normally, nightly runs the 'checkpaths' script to check for
474 discrepancies among the files that list paths to other files, such as
475 exception lists and req.flg. Set this flag to 'n' to disable this
476 check, which appears in the nightly output as "Check lists of files."
477 .RE
478 .LP
479 .B CHECK_DMAKE
480 .RS 5
481 Nightly validates that the version of dmake encountered is known to be
482 safe to use. Set this flag to 'n' to disable this test, allowing any
483 version of dmake to be used.
484 .RE
485 .LP
486 .B MULTI_PROTO
487 .RS 5
488 If "no" (the default),
489 .I nightly
490 will reuse $ROOT for both the DEBUG and non-DEBUG builds. If "yes",
491 the DEBUG build will go in $ROOT and the non-DEBUG build will go in
492 $ROOT-nd. Other values will be treated as "no".
493 .RE
494 .LP
495 .SH NIGHTLY HOOK ENVIRONMENT VARIABLES
496 .LP
497 Several optional environment variables may specify commands to run at
498 various points during the build. Commands specified in the hook
499 variable will be run in a subshell; command output will be appended to
500 the mail message and log file. If the hook exits with a non-zero
501 status, the build is aborted immediately. Environment variables
502 defined in the environment file will be available.
503 .LP
504 .B SYS_PRE_NIGHTLY
505 .RS 5
506 Run just after the workspace lock is acquired. This is reserved for
507 per-build-machine customizations and should be set only in /etc/nightly.conf
508 .RE
509 .LP
510 .B PRE_NIGHTLY
511 .RS 5
512 Run just after SYS_PRE_NIGHTLY.
513 .RE
514 .LP
515 .B PRE_BRINGOVER
516 .RS 5
517 Run just before bringover is started; not run if no bringover is done.
518 .RE
519 .LP
520 .B POST_BRINGOVER
521 .RS 5
522 Run just after bringover completes; not run if no bringover is done.
523 .RE
524 .LP
525 .B POST_NIGHTLY
526 .RS 5
527 Run after the build completes, with the return status of nightly - one
528 of "Completed", "Interrupted", or "Failed" - available in the
529 environment variable NIGHTLY_STATUS.
530 .RE
531 .LP
532 .B SYS_POST_NIGHTLY
533 .RS 5
534 This is reserved for per-build-machine customizations, and runs
535 immediately after POST_NIGHTLY.
536 .RE
537 .LP

```

```

590 .SH REALMODE ENVIRONMENT VARIABLES
591 .LP
592 The following environment variables referenced by
593 .I nightly
594 are only required when the -X option is used.
595 .LP
596 .RE
597 .B IA32_IHV_WS
598 .RS 5
599 Reference to the IHV workspace containing IHV driver binaries.
600 The IHV workspace must be fully built before starting the ON realmode build.
601 .LP
602 .RE
603 .B IA32_IHV_ROOT
604 .RS 5
605 Reference to the IHV workspace proto area.
606 The IHV workspace must be fully built before starting the ON realmode build.
607 .LP
608 .RE
609 .B IA32_IHV_PKGS
610 .RS 5
611 Reference to the IHV workspace packages. If this is empty or the directory
612 is non-existent, then nightly will skip copying the packages.
613 .LP
614 .RE
615 .B IA32_IHV_BINARY_PKGS
616 .RS 5
617 Reference to binary-only IHV packages. If this is empty or the directory
618 is non-existent, then nightly will skip copying the packages.
619 .LP
620 .RE
621 .B SPARC_RM_PKGARCHIVE
622 .RS 5
623 Destination for sparc realmode package SUNWrmodu.
624 Yes, this sparc package really is built on x86.
538 .SH FILES
539 .LP
540 .RS 5
541 /etc/nightly.conf
542 .RE
543 .LP
544 If present, nightly executes this file just prior to executing the
545 .I env
546 file.
634 .SH BUILDING THE IHV WORKSPACE
635 .LP
636 The IHV workspace can be built with
637 .I nightly.
638 The recommended options are:
639 .LP
640 .RS 5
641 NIGHTLY_OPTIONS="-pmWN"
642 .RE
643 .LP
644 None of the realmode environment variables needed for ON realmode builds
645 are required to build the IHV workspace.
547 .SH EXAMPLES
548 .LP
549 Start with the example file in usr/src/tools/env/developer.sh
550 (or gatekeeper.sh), copy to myenv and make your changes.
551 .LP
552 .PD 0
553 # grep NIGHTLY_OPTIONS myenv
554 .LP
555 NIGHTLY_OPTIONS="-ACrlapDm"
556 .LP

```

new/usr/src/tools/scripts/nightly.1

11

```
557 export NIGHTLY_OPTIONS
558 .LP
559 # /opt/onbld/bin/nightly -i myenv
560 .PD
561 .LP
562 .SH SEE ALSO
563 .BR bldenv (1)
```

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

1

```
*****
59019 Sun Jan 26 22:00:01 2014
new/usr/src/tools/scripts/nightly.sh
4526 nightly contains a great deal of effectively dead code
*****
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 (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
25 # Copyright 2008, 2010, Richard Lowe
26 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
27 # Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
28 #
29 # Based on the nightly script from the integration folks,
30 # Mostly modified and owned by mike_s.
31 # Changes also by kjc, dmK.
32 #
33 # BRINGOVER_WS may be specified in the env file.
34 # The default is the old behavior of CLONE_WS
35 #
36 # -i on the command line, means fast options, so when it's on the
37 # command line (only), lint and check builds are skipped no matter what
38 # the setting of their individual flags are in NIGHTLY_OPTIONS.
39 #
40 # LINTDIRS can be set in the env file, format is a list of:
41 #
42 #     /dirname-to-run-lint-on flag
43 #
44 #     Where flag is:  y - enable lint noise diff output
45 #                   n - disable lint noise diff output
46 #
47 #     For example: LINTDIRS="$SRC/uts n $SRC/stand y $SRC/psm y"
48 #
49 # OPTHOME may be set in the environment to override /opt
50 # OPTHOME and TEAMWARE may be set in the environment to override /opt
51 # and /opt/teamware defaults.
52 #
53 #
54 # The CDPATH variable causes ksh's 'cd' builtin to emit messages to stdout
55 # under certain circumstances, which can really screw things up; unset it.
56 #
57 #
58 # Get the absolute path of the nightly script that the user invoked. This
59 # may be a relative path, and we need to do this before changing directory.
```

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

2

```
60 nightly_path='whence $0'
61 #
62 #
63 # Keep track of where we found nightly so we can invoke the matching
64 # which_scm script.  If that doesn't work, don't go guessing, just rely
65 # on the $PATH settings, which will generally give us either /opt/onbld
66 # or the user's workspace.
67 #
68 WHICH_SCM=$(dirname $nightly_path)/which_scm
69 if [[ ! -x $WHICH_SCM ]]; then
70     WHICH_SCM=which_scm
71 fi
72 #
73 function fatal_error
74 {
75     print -u2 "nightly: $"
76     exit 1
77 }
78 #
79 #
80 # Function to do a DEBUG and non-DEBUG build.  Needed because we might
81 # need to do another for the source build, and since we only deliver DEBUG or
82 # non-DEBUG packages.
83 #
84 # usage: normal_build
85 #
86 function normal_build {
87 #
88     typeset orig_p_FLAG="$p_FLAG"
89     typeset crypto_signer="$CODESIGN_USER"
90 #
91     suffix=""
92 #
93     # non-DEBUG build begins
94 #
95     if [ "$F_FLAG" = "n" ]; then
96         set_non_debug_build_flags
97         CODESIGN_USER="$crypto_signer" \
98         build "non-DEBUG" "$suffix-nd" "-nd" "$MULTI_PROTO"
99     else
100         if [ "$build_ok" = "y" -a "$X_FLAG" = "y" -a \
101             "$p_FLAG" = "y" ]; then
102             copy_ihv_pkgs non-DEBUG -nd
103         fi
104     fi
105     echo "\n=== No non-DEBUG $open_only build ===\n" >> "$LOGFILE"
106 #
107     # non-DEBUG build ends
108 #
109     # DEBUG build begins
110 #
111     if [ "$D_FLAG" = "y" ]; then
112         set_debug_build_flags
113         CODESIGN_USER="$crypto_signer" \
114         build "DEBUG" "$suffix" "" "$MULTI_PROTO"
115     else
116         if [ "$build_ok" = "y" -a "$X_FLAG" = "y" -a \
117             "$p_FLAG" = "y" ]; then
118             copy_ihv_pkgs DEBUG ""
119         fi
120     fi
121     echo "\n=== No DEBUG $open_only build ===\n" >> "$LOGFILE"
122 #
123     # DEBUG build ends
124 #
125     p_FLAG="$orig_p_FLAG"
126 }
```

```

118 }
    unchanged_portion_omitted
159 #
160 # usage: filelist DESTDIR PATTERN
161 #
162 function filelist {
163     DEST=$1
164     PATTERN=$2
165     cd ${DEST}
166 }

168 # function to save off binaries after a full build for later
169 # restoration
170 function save_binaries {
171     # save off list of binaries
172     echo "\n=== Saving binaries from build at 'date' ===\n" | \
173     tee -a $mail_msg_file >> $LOGFILE
174     rm -f ${BINARCHIVE}
175     cd ${CODEMGR_WS}
176     filelist ${CODEMGR_WS} '^preserve:' >> $LOGFILE
177     filelist ${CODEMGR_WS} '^preserve:' | \
178     cpio -ocB 2>/dev/null | compress \
179     > ${BINARCHIVE}
180 }

182 # delete files
183 # usage: hybridize_files DESTDIR MAKE_TARGET
184 function hybridize_files {
185     DEST=$1
186     MAKETARG=$2

188     echo "\n=== Hybridizing files at 'date' ===\n" | \
189     tee -a $mail_msg_file >> $LOGFILE
190     for i in `filelist ${DEST} '^delete:'`
191     do
192         echo "removing ${i}." | tee -a $mail_msg_file >> $LOGFILE
193         rm -rf "${i}"
194     done
195     for i in `filelist ${DEST} '^hybridize:'`
196     do
197         echo "hybridizing ${i}." | tee -a $mail_msg_file >> $LOGFILE
198         rm -f ${i}+
199         sed -e "/^# HYBRID DELETE START/,/^# HYBRID DELETE END/d" \
200         < ${i} > ${i}+
201         mv ${i}+ ${i}
202     done
203 }

205 # restore binaries into the proper source tree.
206 # usage: restore_binaries DESTDIR MAKE_TARGET
207 function restore_binaries {
208     DEST=$1
209     MAKETARG=$2

211     echo "\n=== Restoring binaries to ${MAKETARG} at 'date' ===\n" | \
212     tee -a $mail_msg_file >> $LOGFILE
213     cd ${DEST}
214     zcat ${BINARCHIVE} | \
215     cpio -idmucvB 2>/dev/null | tee -a $mail_msg_file >> ${LOGFILE}
216 }

218 # rename files we save binaries of
219 # usage: rename_files DESTDIR MAKE_TARGET
220 function rename_files {
221     DEST=$1

```

```

222     MAKETARG=$2
223     echo "\n=== Renaming source files in ${MAKETARG} at 'date' ===\n" | \
224     tee -a $mail_msg_file >> $LOGFILE
225     for i in `filelist ${DEST} '^rename:'`
226     do
227         echo ${i} | tee -a $mail_msg_file >> ${LOGFILE}
228         rm -f ${i}.export
229         mv ${i} ${i}.export
230     done
231 }

150 # Return library search directive as function of given root.
151 function myldlibs {
152     echo "-L${1}/lib -L${1}/usr/lib"
153 }
    unchanged_portion_omitted

160 #
161 # Function to do the build, including package generation.
162 # usage: build LABEL SUFFIX ND MULTIPROTO
163 # - LABEL is used to tag build output.
164 # - SUFFIX is used to distinguish files (e.g., DEBUG vs non-DEBUG,
165 #   open-only vs full tree).
166 # - ND is "-nd" (non-DEBUG builds) or "" (DEBUG builds).
167 # - If MULTIPROTO is "yes", it means to name the proto area according to
168 #   SUFFIX. Otherwise ("no"), (re)use the standard proto area.
169 #
170 function build {
171     LABEL=$1
172     SUFFIX=$2
173     ND=$3
174     MULTIPROTO=$4
175     INSTALLOG=install${SUFFIX}-${MACH}
176     NOISE=noise${SUFFIX}-${MACH}
177     PKGARCHIVE=${PKGARCHIVE_ORIG}${SUFFIX}

179     ORIGROOT=$ROOT
180     [ $MULTIPROTO = no ] || export ROOT=$ROOT${SUFFIX}

182     export ENVLDLIBS1='myldlibs $ROOT'
183     export ENVCPPFLAGS1='myheaders $ROOT'

185     this_build_ok=y
186     #
187     #   Build OS-Networking source
188     #
189     echo "\n=== Building OS-Net source at 'date' ($LABEL) ===\n" \
190     >> $LOGFILE

192     rm -f $SRC/${INSTALLOG}.out
193     cd $SRC
194     /bin/time $MAKE -e install 2>&1 | \
195     tee -a $SRC/${INSTALLOG}.out >> $LOGFILE

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

197     echo "\n=== Build errors ($LABEL) ===\n" >> $mail_msg_file
198     grep ":" $SRC/${INSTALLOG}.out |
199     grep -e "(^${MAKE}:|[ lerror[: \n])" | \
200     grep -v "Ignoring unknown host" | \
201     grep -v "cc .* -o error " | \
202     grep -v "warning" >> $mail_msg_file

```

```

203     if [ "$?" = "0" ]; then
204         build_ok=n
205         this_build_ok=n
206     fi
207     grep "bootblock image is .* bytes too big" $SRC/${INSTALLOG}.out \
208     >> $mail_msg_file
209     if [ "$?" = "0" ]; then
210         build_ok=n
211         this_build_ok=n
212     fi

303     if [ "$W_FLAG" = "n" ]; then
214     echo "\n==== Build warnings ($LABEL) ====\n" >>$mail_msg_file
215     egrep -i warning: $SRC/${INSTALLOG}.out \
216     | egrep -v '^tic:' \
217     | egrep -v "symbol (\|'|)timezone' has differing types:" \
218     | egrep -v "parameter <PSTAMP> set to" \
219     | egrep -v "Ignoring unknown host" \
220     | egrep -v "redefining segment flags attribute for" \
221     >> $mail_msg_file
212     fi

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

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

229     if [ "$i_FLAG" = "n" ]; then
320     if [ "$i_FLAG" = "n" -a "$W_FLAG" = "n" ]; then
230         rm -f $SRC/${NOISE}.ref
231         if [ -f $SRC/${NOISE}.out ]; then
232             mv $SRC/${NOISE}.out $SRC/${NOISE}.ref
233         fi
234         grep : $SRC/${INSTALLOG}.out \
235         | egrep -v '^/' \
236         | egrep -v '^((Start|Finish|real|user|sys|./bld_awk)' \
237         | egrep -v '^tic:' \
238         | egrep -v '^mcs' \
239         | egrep -v '^LD_LIBRARY_PATH=' \
240         | egrep -v 'ar: creating' \
241         | egrep -v 'ar: writing' \
242         | egrep -v 'conflicts:' \
243         | egrep -v ':saved created' \
244         | egrep -v '^stty.*c:' \
245         | egrep -v '^mfname.c:' \
246         | egrep -v '^uname-i.c:' \
247         | egrep -v '^volumes.c:' \
248         | egrep -v '^lint library construction:' \
249         | egrep -v 'tsort: INFORM:' \
250         | egrep -v 'stripalign:' \
251         | egrep -v 'chars, width' \
252         | egrep -v "symbol (\|'|)timezone' has differing types:" \
253         | egrep -v 'PSTAMP' \
254         | egrep -v '|%WHOANDWHERE%|' \
255         | egrep -v '^Manifesting' \
256         | egrep -v 'Ignoring unknown host' \
257         | egrep -v 'Processing method:' \
258         | egrep -v '^Writing' \
259         | egrep -v '^spellinl:' \
260         | egrep -v '^adding:' \
261         | egrep -v '^echo 'msgid' \
262         | egrep -v '^echo ' \
263         | egrep -v '\\.c:$' \
264         | egrep -v '^Adding file:' \
265         | egrep -v 'CLASSPATH=' \

```

```

266     | egrep -v '\var/mail/:saved' \
267     | egrep -v -- '-DUTS_VERSION=' \
268     | egrep -v '^Running Mkbootstrap' \
269     | egrep -v '^Applet length read:' \
270     | egrep -v 'bytes written:' \
271     | egrep -v '^File:SolarisAuthApplet.bin' \
272     | egrep -v -i 'jibversion' \
273     | egrep -v '^Output size:' \
274     | egrep -v '^Solo size statistics:' \
275     | egrep -v '^Using ROM API Version' \
276     | egrep -v '^Zero Signature length:' \
277     | egrep -v '^Note \ (probably harmless\):' \
278     | egrep -v ':' \
279     | egrep -v -- '-xcache' \
280     | egrep -v '^+' \
281     | egrep -v '^ccl: note: -fwritable-strings' \
282     | egrep -v 'svccfg-native -s svc:/' \
283     | sort | uniq >$SRC/${NOISE}.out
284     if [ ! -f $SRC/${NOISE}.ref ]; then
285         cp $SRC/${NOISE}.out $SRC/${NOISE}.ref
286     fi
287     echo "\n==== Build noise differences ($LABEL) ====\n" \
288     >>$mail_msg_file
289     diff $SRC/${NOISE}.ref $SRC/${NOISE}.out >>$mail_msg_file
290     fi

292     #
293     # Re-sign selected binaries using signing server
294     # (gatekeeper builds only)
295     #
296     if [ -n "$CODESIGN_USER" -a "$this_build_ok" = "y" ]; then
297         echo "\n==== Signing proto area at 'date' ====\n" >> $LOGFILE
298         signing_file=${TMPDIR}/signing
299         rm -f ${signing_file}
300         export CODESIGN_USER
301         signproto $SRC/tools/codesign/creds 2>&1 | \
302         tee -a ${signing_file} >> $LOGFILE
303         echo "\n==== Finished signing proto area at 'date' ====\n" \
304         >> $LOGFILE
305         echo "\n==== Crypto module signing errors ($LABEL) ====\n" \
306         >> $mail_msg_file
307         egrep 'WARNING|ERROR' ${signing_file} >> $mail_msg_file
308         if (( $? == 0 )) ; then
309             build_ok=n
310             this_build_ok=n
311         fi
312     fi

314     #
315     # Building Packages
316     #
317     if [ "$p_FLAG" = "y" -a "$this_build_ok" = "y" ]; then
318         if [ -d $SRC/pkg ]; then
409         if [ -d $SRC/pkg -o -d $SRC/pkgdefs ]; then
319             echo "\n==== Creating $LABEL packages at 'date' ====\n"
320             >> $LOGFILE
321             echo "Clearing out $PKGARCHIVE ..." >> $LOGFILE
322             rm -rf $PKGARCHIVE >> "$LOGFILE" 2>&1
323             mkdir -p $PKGARCHIVE >> "$LOGFILE" 2>&1

325             rm -f $SRC/pkg/${INSTALLOG}.out
326             cd $SRC/pkg
416             for d in pkg pkgdefs; do
417                 if [ ! -f "$SRC/$d/Makefile" ]; then
418                     continue
419                 fi

```

```

420         rm -f $SRC/$d/${INSTALLOG}.out
421         cd $SRC/$d
327         /bin/time $MAKE -e install 2>&1 | \
328         tee -a $SRC/pkg/${INSTALLOG}.out >> $LOGFILE
423         tee -a $SRC/$d/${INSTALLOG}.out >> $LOGF
424         done
330         echo "\n==== package build errors ($LABEL) ==== \n" \
331         >> $mail_msg_file
333         egrep "${MAKE}|ERROR|WARNING" $SRC/pkg/${INSTALLOG}.out
429         for d in pkg pkgdefs; do
430             if [ ! -f "$SRC/$d/Makefile" ]; then
431                 continue
432             fi
434             egrep "${MAKE}|ERROR|WARNING" $SRC/$d/${INSTALLO
435             grep ':' | \
436             grep -v PSTAMP | \
437             egrep -v "Ignoring unknown host" \
438             >> $mail_msg_file
439         done
338         else
339             #
340             # Handle it gracefully if -p was set but there are
341             # neither pkg directories.
443             # neither pkg nor pkgdefs directories.
342             #
343             echo "\n==== No $LABEL packages to build ==== \n" \
344             >> $LOGFILE
345         fi
346     else
347         echo "\n==== Not creating $LABEL packages ==== \n" >> $LOGFILE
348     fi
350     ROOT=$ORIGROOT
351 }

```

unchanged_portion_omitted

```

539 # Install proto area from IHV build
541 function copy_ihv_proto {
543     echo "\n==== Installing IHV proto area ==== \n" \
544     >> $LOGFILE
545     if [ -d "$IA32_IHV_ROOT" ]; then
546         if [ ! -d "$ROOT" ]; then
547             echo "mkdir -p $ROOT" >> $LOGFILE
548             mkdir -p $ROOT
549         fi
550         echo "copying $IA32_IHV_ROOT to $ROOT \n" >> $LOGFILE
551         cd $IA32_IHV_ROOT
552         tar cf - . | (cd $ROOT; umask 0; tar xpf - ) 2>&1 >> $LOGFILE
553     else
554         echo "$IA32_IHV_ROOT: not found" >> $LOGFILE
555     fi
557     if [ "$MULTI_PROTO" = yes ]; then
558         if [ ! -d "$ROOT-nd" ]; then
559             echo "mkdir -p $ROOT-nd" >> $LOGFILE
560             mkdir -p $ROOT-nd
561         fi
562         # If there's a non-DEBUG version of the IHV proto area,
563         # copy it, but copy something if there's not.
564         if [ -d "$IA32_IHV_ROOT-nd" ]; then
565             echo "copying $IA32_IHV_ROOT-nd to $ROOT-nd \n" >> $LOGFI

```

```

566         cd $IA32_IHV_ROOT-nd
567     elif [ -d "$IA32_IHV_ROOT" ]; then
568         echo "copying $IA32_IHV_ROOT to $ROOT-nd \n" >> $LOGFILE
569         cd $IA32_IHV_ROOT
570     else
571         echo "$IA32_IHV_ROOT{-nd,}: not found" >> $LOGFILE
572     fi
573     return
574     tar cf - . | (cd $ROOT-nd; umask 0; tar xpf - ) 2>&1 >> $LOGFILE
575 fi
576 }
578 # Install IHV packages in PKGARCHIVE
579 # usage: copy_ihv_pkgs LABEL SUFFIX
580 function copy_ihv_pkgs {
581     LABEL=$1
582     SUFFIX=$2
583     # always use non-DEBUG IHV packages
584     IA32_IHV_PKGS=${IA32_IHV_PKGS_ORIG}-nd
585     PKGARCHIVE=${PKGARCHIVE_ORIG}${SUFFIX}
587     echo "\n==== Installing IHV packages from $IA32_IHV_PKGS ($LABEL) ==== \n"
588     >> $LOGFILE
589     if [ -d "$IA32_IHV_PKGS" ]; then
590         cd $IA32_IHV_PKGS
591         tar cf - * | \
592         (cd $PKGARCHIVE; umask 0; tar xpf - ) 2>&1 >> $LOGFILE
593     else
594         echo "$IA32_IHV_PKGS: not found" >> $LOGFILE
595     fi
597     echo "\n==== Installing IHV packages from $IA32_IHV_BINARY_PKGS ($LABEL)
598     >> $LOGFILE
599     if [ -d "$IA32_IHV_BINARY_PKGS" ]; then
600         cd $IA32_IHV_BINARY_PKGS
601         tar cf - * | \
602         (cd $PKGARCHIVE; umask 0; tar xpf - ) 2>&1 >> $LOGFILE
603     else
604         echo "$IA32_IHV_BINARY_PKGS: not found" >> $LOGFILE
605     fi
606 }
437 #
438 # Build and install the onbld tools.
439 #
440 # usage: build_tools DESTROOT
441 #
442 # returns non-zero status if the build was successful.
443 #
444 function build_tools {
445     DESTROOT=$1
447     INSTALLOG=install-{$MACH}
449     echo "\n==== Building tools at 'date' ==== \n" \
450     >> $LOGFILE
452     rm -f ${TOOLS}/${INSTALLOG}.out
453     cd ${TOOLS}
454     /bin/time $MAKE TOOLS_PROTO=${DESTROOT} -e install 2>&1 | \
455     tee -a ${TOOLS}/${INSTALLOG}.out >> $LOGFILE
457     echo "\n==== Tools build errors ==== \n" >> $mail_msg_file
459     egrep ":" ${TOOLS}/${INSTALLOG}.out |
460     egrep -e "${MAKE}:|[ lerror[: \n])" | \

```

```

461         egrep -v "Ignoring unknown host" | \
462         egrep -v warning >> $mail_msg_file
463     return $?
464 }
unchanged portion omitted

550 #
551 # Verify that the closed bins are present
552 # Verify that the closed tree is present if it needs to be.
553 function check_closed_bins {
554 function check_closed_tree {
555     if [[ ! -d "$ON_CLOSED_BINS" ]]; then
556         echo "ON_CLOSED_BINS must point to the closed binaries tree."
557         build_ok=n
558         exit 1
559     fi
560 }
561 }
562 function obsolete_build {
563     echo "WARNING: Obsolete $1 build requested; request will be ignored"
564 }
565 }
566 # wrapper over wsdiff.
567 # usage: do_wsdiff LABEL OLDPROTO NEWPROTO
568 #
569 function do_wsdiff {
570     label=$1
571     oldproto=$2
572     newproto=$3
573
574     wsdiff="wsdiff"
575     [ "$t_FLAG" = y ] && wsdiff="wsdiff -t"
576
577     echo "\n==== Getting object changes since last build at 'date' \
578     "($label) ====\n" | tee -a $LOGFILE >> $mail_msg_file
579     $wsdiff -s -r ${TMPDIR}/wsdiff.results $oldproto $newproto 2>&1 | \
580     tee -a $LOGFILE >> $mail_msg_file
581     echo "\n==== Object changes determined at 'date' ($label) ====\n" | \
582     tee -a $LOGFILE >> $mail_msg_file
583 }
unchanged portion omitted

599 MACH=`uname -p`

600 if [ "$OPTHOME" = "" ]; then
601     OPTHOME=/opt
602     export OPTHOME
603 fi
604 if [ "$TEAMWARE" = "" ]; then
605     TEAMWARE=$OPTHOME/teamware
606     export TEAMWARE
607 fi

608 USAGE='Usage: nightly [-in] [+t] [-V VERS ] <env_file>'

609 Where:
610 -i      Fast incremental options (no clobber, lint, check)
611 -n      Do not do a bringover
612 +t      Use the build tools in $ONBLD_TOOLS/bin
613 -V VERS set the build version string to VERS

614 <env_file> file in Bourne shell syntax that sets and exports
615 variables that configure the operation of this script and many of

```

```

616 the scripts this one calls. If <env_file> does not exist,
617 it will be looked for in $OPTHOME/onbld/env.

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

620 -A      check for ABI differences in .so files
621 -C      check for cstyle/hdrchk errors
622 -D      do a build with DEBUG on
623 -F      do _not_ do a non-DEBUG build
624 -G      gate keeper default group of options (-au)
625 -I      integration engineer default group of options (-ampu)
626 -M      do not run pmodes (safe file permission checker)
627 -N      do not run protocmp
628 -R      default group of options for building a release (-mp)
629 -U      update proto area in the parent
630 -V VERS set the build version string to VERS
631 -X      copy x86 IHV proto area
632 -f      find unreferenced files
633 -i      do an incremental build (no "make clobber")
634 -l      do "make lint" in $LINTDIRS (default: $SRC y)
635 -m      send mail to $MAILTO at end of build
636 -n      do not do a bringover
637 -o      build using root privileges to set OWNER/GROUP (old style)
638 -p      create packages
639 -r      check ELF runtime attributes in the proto area
640 -t      build and use the tools in $SRC/tools (default setting)
641 +t      Use the build tools in $ONBLD_TOOLS/bin
642 -u      update proto_list_$MACH and friends in the parent workspace;
643         when used with -f, also build an unrefmaster.out in the parent
644 -w      report on differences between previous and current proto areas
645 -z      compress cpio archives with gzip
646 -W      Do not report warnings (freeware gate ONLY)
647 #
648 # A log file will be generated under the name $LOGFILE
649 # for partially completed build and log.'date +%F'
650 # in the same directory for fully completed builds.

651 # default values for low-level FLAGS; G I R are group FLAGS
652 A_FLAG=n
653 C_FLAG=n
654 D_FLAG=n
655 F_FLAG=n
656 f_FLAG=n
657 i_FLAG=n; i_CMD_LINE_FLAG=n
658 l_FLAG=n
659 M_FLAG=n
660 m_FLAG=n
661 N_FLAG=n
662 n_FLAG=n
663 o_FLAG=n
664 P_FLAG=n
665 p_FLAG=n
666 r_FLAG=n
667 T_FLAG=n
668 t_FLAG=y
669 U_FLAG=n
670 u_FLAG=n
671 V_FLAG=n
672 W_FLAG=n
673 w_FLAG=n
674 X_FLAG=n
675 #
676 XMOD_OPT=

```



```

671 #
672 build_ok=y

674 #
675 # examine arguments
676 #

678 OPTIND=1
679 while getopts +intV: FLAG
680 do
681     case $FLAG in
682         i ) i_FLAG=y; i_CMD_LINE_FLAG=y
683             ;;
684         n ) n_FLAG=y
685             ;;
686         +t ) t_FLAG=n
687             ;;
688         V ) V_FLAG=y
689             V_ARG="$OPTARG"
690             ;;
691         \? ) echo "$USAGE"
692             exit 1
693             ;;
694     esac
695 done

697 # correct argument count after options
698 shift `expr $OPTIND - 1`

700 # test that the path to the environment-setting file was given
701 if [ $# -ne 1 ]; then
702     echo "$USAGE"
703     exit 1
704 fi

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

712 #
713 # force locale to C
714 LC_COLLATE=C; export LC_COLLATE
715 LC_CTYPE=C; export LC_CTYPE
716 LC_MESSAGES=C; export LC_MESSAGES
717 LC_MONETARY=C; export LC_MONETARY
718 LC_NUMERIC=C; export LC_NUMERIC
719 LC_TIME=C; export LC_TIME

721 # clear environment variables we know to be bad for the build
722 unset LD_OPTIONS
723 unset LD_AUDIT LD_AUDIT_32 LD_AUDIT_64
724 unset LD_BIND_NOW LD_BIND_NOW_32 LD_BIND_NOW_64
725 unset LD_BREADTH LD_BREADTH_32 LD_BREADTH_64
726 unset LD_CONFIG LD_CONFIG_32 LD_CONFIG_64
727 unset LD_DEBUG LD_DEBUG_32 LD_DEBUG_64
728 unset LD_DEMANGLE LD_DEMANGLE_32 LD_DEMANGLE_64
729 unset LD_FLAGS LD_FLAGS_32 LD_FLAGS_64
730 unset LD_LIBRARY_PATH LD_LIBRARY_PATH_32 LD_LIBRARY_PATH_64
731 unset LD_LOADFLTR LD_LOADFLTR_32 LD_LOADFLTR_64
732 unset LD_NOAUDIT LD_NOAUDIT_32 LD_NOAUDIT_64
733 unset LD_NOAUXFLTR LD_NOAUXFLTR_32 LD_NOAUXFLTR_64
734 unset LD_NOCONFIGN LD_NOCONFIGN_32 LD_NOCONFIGN_64
735 unset LD_NODIRCONFIG LD_NODIRCONFIG_32 LD_NODIRCONFIG_64
736 unset LD_NODIRECT LD_NODIRECT_32 LD_NODIRECT_64

```

```

737 unset LD_NOLAZYLOAD LD_NOLAZYLOAD_32 LD_NOLAZYLOAD_64
738 unset LD_NOOBJALTER LD_NOOBJALTER_32 LD_NOOBJALTER_64
739 unset LD_NOVERSION LD_NOVERSION_32 LD_NOVERSION_64
740 unset LD_ORIGIN LD_ORIGIN_32 LD_ORIGIN_64
741 unset LD_PRELOAD LD_PRELOAD_32 LD_PRELOAD_64
742 unset LD_PROFILE LD_PROFILE_32 LD_PROFILE_64

744 unset CONFIG
745 unset GROUP
746 unset OWNER
747 unset REMOTE
748 unset ENV
749 unset ARCH
750 unset CLASSPATH
751 unset NAME

753 #
754 # To get ONBLD_TOOLS from the environment, it must come from the env file.
755 # If it comes interactively, it is generally TOOLS_PROTO, which will be
756 # clobbered before the compiler version checks, which will therefore fail.
757 #
758 unset ONBLD_TOOLS

760 #
761 # Setup environmental variables
762 #
763 if [ -f /etc/nightly.conf ]; then
764     . /etc/nightly.conf
765 fi

767 if [ -f $1 ]; then
768     if [[ $1 = */* ]]; then
769         . $1
770     else
771         . ./$1
772     fi
773 else
774     if [ -f $OPTHOME/onbld/env/$1 ]; then
775         . $OPTHOME/onbld/env/$1
776     else
777         echo "Cannot find env file as either $1 or $OPTHOME/onbld/env/$1"
778         exit 1
779     fi
780 fi

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

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

796 #
797 # place ourselves in a new task, respecting BUILD_PROJECT if set.
798 #
799 if [ -z "$BUILD_PROJECT" ]; then
800     /usr/bin/newtask -c $$
801 else
802     /usr/bin/newtask -c $$ -p $BUILD_PROJECT

```

```

803 fi

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

808 #
809 # See if NIGHTLY_OPTIONS is set
810 #
811 if [ "$NIGHTLY_OPTIONS" = "" ]; then
812     NIGHTLY_OPTIONS="-aBm"
813 fi

815 #
816 # If BRINGOVER_WS was not specified, let it default to CLONE_WS
817 #
818 if [ "$BRINGOVER_WS" = "" ]; then
819     BRINGOVER_WS=$CLONE_WS
820 fi

822 #
1013 # If CLOSED_BRINGOVER_WS was not specified, let it default to CLOSED_CLONE_WS
1014 #
1015 if [ "$CLOSED_BRINGOVER_WS" = "" ]; then
1016     CLOSED_BRINGOVER_WS=$CLOSED_CLONE_WS
1017 fi

1019 #
823 # If BRINGOVER_FILES was not specified, default to usr
824 #
825 if [ "$BRINGOVER_FILES" = "" ]; then
826     BRINGOVER_FILES="usr"
827 fi

829 check_closed_bins
1026 check_closed_tree

831 #
832 # Note: changes to the option letters here should also be applied to the
833 #       bldenv script. 'd' is listed for backward compatibility.
834 #
835 NIGHTLY_OPTIONS=-${NIGHTLY_OPTIONS#-}
836 OPTIND=1
837 while getopts +ABCDdFfGIilMmNnpRrtUuw FLAG $NIGHTLY_OPTIONS
1034 while getopts +ABCDdFfGIilMmNnoPpRrTtUuWwXxz FLAG $NIGHTLY_OPTIONS
838 do
839     case $FLAG in
840         A ) A_FLAG=y
841             ;;
842         B ) D_FLAG=y
843             ;; # old version of D
844         C ) C_FLAG=y
845             ;;
846         D ) D_FLAG=y
847             ;;
848         F ) F_FLAG=y
849             ;;
850         f ) f_FLAG=y
851             ;;
852         G ) u_FLAG=y
853             ;;
854         I ) m_FLAG=y
855             p_FLAG=y
856             u_FLAG=y
857             ;;
858         i ) i_FLAG=y
859             ;;

```

```

860     l ) l_FLAG=y
861         ;;
862     M ) M_FLAG=y
863         ;;
864     m ) m_FLAG=y
865         ;;
866     N ) N_FLAG=y
867         ;;
868     n ) n_FLAG=y
869         ;;
1067     o ) o_FLAG=y
1068         ;;
1069     P ) P_FLAG=y
1070         ;; # obsolete
870     p ) p_FLAG=y
871         ;;
872     R ) m_FLAG=y
873         p_FLAG=y
874         ;;
875     r ) r_FLAG=y
876         ;;
1078     T ) T_FLAG=y
1079         ;; # obsolete
877     +t ) t_FLAG=n
878         ;;
879     U ) if [ -z "${PARENT_ROOT}" ]; then
880         echo "PARENT_ROOT must be set if the U flag is" \
881             "present in NIGHTLY_OPTIONS."
882         exit 1
883     fi
884     NIGHTLY_PARENT_ROOT=$PARENT_ROOT
885     if [ -n "${PARENT_TOOLS_ROOT}" ]; then
886         NIGHTLY_PARENT_TOOLS_ROOT=$PARENT_TOOLS_ROOT
887     fi
888     U_FLAG=y
889     ;;
890     u ) u_FLAG=y
891         ;;
1095     W ) W_FLAG=y
1096         ;;

892     w ) w_FLAG=y
893         ;;
1100     X ) # now that we no longer need realmode builds, just
1101         # copy IHV packages. only meaningful on x86.
1102         if [ "$MACH" = "i386" ]; then
1103             X_FLAG=y
1104         fi
1105         ;;
1106     x ) XMOD_OPT="-x"
1107         ;;
894     \? ) echo "$USAGE"
895         exit 1
896         ;;
897     esac
898 done

900 if [ $ISUSER -ne 0 ]; then
1115     if [ "$o_FLAG" = "y" ]; then
1116         echo "Old-style build requires root permission."
1117         exit 1
1118     fi

901     # Set default value for STAFFER, if needed.
902     if [ -z "$STAFFER" -o "$STAFFER" = "nobody" ]; then
903         STAFFER="/usr/xpg4/bin/id -un"

```

```

904         export STAFFER
905     fi
906 fi

908 if [ -z "$MAILTO" -o "$MAILTO" = "nobody" ]; then
909     MAILTO=$STAFFER
910     export MAILTO
911 fi

913 PATH="$OPTHOME/onbld/bin:$OPTHOME/onbld/bin/${MACH}:/usr/ccs/bin"
914 PATH="$PATH:$OPTHOME/SUNWspro/bin:/usr/bin:/usr/sbin:/usr/ucb"
1133 PATH="$PATH:$OPTHOME/SUNWspro/bin:$TEAMWARE/bin:/usr/bin:/usr/sbin:/usr/ucb"
915 PATH="$PATH:/usr/openwin/bin:/usr/sfw/bin:/opt/sfw/bin:."
916 export PATH

918 # roots of source trees, both relative to $SRC and absolute.
919 relsrcdirs="."
920 absrsrcdirs="$SRC"

922 PROTOCMPTERSE="protocmp.terse -gu"
1141 unset CH
1142 if [ "$o_FLAG" = "y" ]; then
1143     # root invoked old-style build -- make sure it works as it always has
1144     # by exporting 'CH'. The current Makefile.master doesn't use this, but
1145     # the old ones still do.
1146     PROTOCMPTERSE="protocmp.terse"
1147     CH=
1148     export CH
1149 else
1150     PROTOCMPTERSE="protocmp.terse -gu"
1151 fi
923 POUND_SIGN="#"
924 # have we set RELEASE_DATE in our env file?
925 if [ -z "$RELEASE_DATE" ]; then
926     RELEASE_DATE=$(LC_ALL=C date +"%B %Y")
927 fi
928 BUILD_DATE=$(LC_ALL=C date +%Y-%b-%d)
929 BASEWSDIR=$(basename $CODEMGR_WS)
930 DEV_CM="\@(#)SunOS Internal Development: $LOGNAME $BUILD_DATE [$BASEWSDIR]\\""

932 # we export POUND_SIGN, RELEASE_DATE and DEV_CM to speed up the build process
933 # by avoiding repeated shell invocations to evaluate Makefile.master
934 # definitions.
935 export POUND_SIGN RELEASE_DATE DEV_CM
1162 # by avoiding repeated shell invocations to evaluate Makefile.master definitions
1163 # we export o_FLAG and X_FLAG for use by makebfu, and by usr/src/pkg/Makefile
1164 export o_FLAG X_FLAG POUND_SIGN RELEASE_DATE DEV_CM

937 maketype="distributed"
938 if [[ -z "$MAKE" ]]; then
939     MAKE=dmake
940 elif [[ ! -x "$MAKE" ]]; then
941     echo "\$MAKE is set to garbage in the environment"
942     exit 1
943 fi
944 # get the dmake version string alone
945 DMAKE_VERSION=$( $MAKE -v )
946 DMAKE_VERSION=${DMAKE_VERSION#*: }
947 # focus in on just the dotted version number alone
948 DMAKE_MAJOR=$( echo $DMAKE_VERSION | \
949     sed -e 's/.*\<([^\.]*)\.[^\.]*/\1/' )
950 # extract the second (or final) integer
951 DMAKE_MINOR=${DMAKE_MAJOR#*.}
952 DMAKE_MINOR=${DMAKE_MINOR%*.}
953 # extract the first integer
954 DMAKE_MAJOR=${DMAKE_MAJOR%*.}

```

```

955 CHECK_DMAKE=${CHECK_DMAKE:-y}
956 # x86 was built on the 12th, sparc on the 13th.
957 if [ "$CHECK_DMAKE" = "y" -a \
958     "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/12" -a \
959     "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/13" -a \
960     "$DMAKE_MAJOR" -lt 7 -o \
961     "$DMAKE_MAJOR" -eq 7 -a "$DMAKE_MINOR" -lt 4 \ ] ]; then
962     if [ -z "$DMAKE_VERSION" ]; then
963         echo "DMAKE is missing."
964         exit 1
965     fi
966     echo 'whence $MAKE\' version is:"
967     echo " ${DMAKE_VERSION}"
968     cat <<EOF

970 This version may not be safe for use, if you really want to use this version
971 anyway add the following to your environment to disable this check:
1199 This version may not be safe for use. Either set TEAMWARE to a better
1200 path or (if you really want to use this version of dmake anyway), add
1201 the following to your environment to disable this check:

973     CHECK_DMAKE=n
974 EOF
975     exit 1
976 fi
977 export PATH
978 export MAKE

980 if [ "${SUNWSPRO}" != "" ]; then
981     PATH="${SUNWSPRO}/bin:$PATH"
982     export PATH
983 fi

985 hostname=$(uname -n)
986 if [[ $DMAKE_MAX_JOBS != +([0-9]) || $DMAKE_MAX_JOBS -eq 0 ]]
987 then
988     maxjobs=
989     if [[ -f $HOME/.make.machines ]]
990     then
991         # Note: there is a hard tab and space character in the [ ]s
992         # below.
993         egrep -i "^[ \t]*$hostname[ \t]*\ \"
994             $HOME/.make.machines | read host jobs
995         maxjobs=${jobs##*=}
996     fi

998     if [[ $maxjobs != +([0-9]) || $maxjobs -eq 0 ]]
999     then
1000         # default
1001         maxjobs=4
1002     fi

1004     export DMAKE_MAX_JOBS=$maxjobs
1005 fi

1007 DMAKE_MODE=parallel;
1008 export DMAKE_MODE

1010 if [ -z "${ROOT}" ]; then
1011     echo "ROOT must be set."
1012     exit 1
1013 fi

1015 #
1016 # if -V flag was given, reset VERSION to V_ARG
1017 #

```

```

1018 if [ "$V_FLAG" = "y" ]; then
1019     VERSION=$V_ARG
1020 fi

1252 #
1253 # Check for IHV root for copying ihv proto area
1254 #
1255 if [ "$X_FLAG" = "y" ]; then
1256     if [ "$IA32_IHV_ROOT" = "" ]; then
1257         echo "IA32_IHV_ROOT: must be set for copying ihv proto"
1258         args_ok=n
1259     fi
1260     if [ ! -d "$IA32_IHV_ROOT" ]; then
1261         echo "IA32_IHV_ROOT: not found"
1262         args_ok=n
1263     fi
1264     if [ "$IA32_IHV_WS" = "" ]; then
1265         echo "IA32_IHV_WS: must be set for copying ihv proto"
1266         args_ok=n
1267     fi
1268     if [ ! -d "$IA32_IHV_WS" ]; then
1269         echo "IA32_IHV_WS: not found"
1270         args_ok=n
1271     fi
1272 fi

1022 TMPDIR="/tmp/nightly.tmpdir.$$"
1023 export TMPDIR
1024 rm -rf ${TMPDIR}
1025 mkdir -p $TMPDIR || exit 1
1026 chmod 777 $TMPDIR

1028 #
1029 # Keep elfsign's use of pkcs11_softtoken from looking in the user home
1030 # directory, which doesn't always work. Needed until all build machines
1031 # have the fix for 6271754
1032 #
1033 SOFTTOKEN_DIR=$TMPDIR
1034 export SOFTTOKEN_DIR

1036 #
1037 # Tools should only be built non-DEBUG. Keep track of the tools proto
1038 # area path relative to $TOOLS, because the latter changes in an
1039 # export build.
1040 #
1041 # TOOLS_PROTO is included below for builds other than usr/src/tools
1042 # that look for this location. For usr/src/tools, this will be
1043 # overridden on the $MAKE command line in build_tools().
1044 #
1045 TOOLS=${SRC}/tools
1046 TOOLS_PROTO_REL=proto/root_${MACH}-nd
1047 TOOLS_PROTO=${TOOLS}/${TOOLS_PROTO_REL}; export TOOLS_PROTO

1049 unset CFLAGS LD_LIBRARY_PATH LDFLAGS

1051 # create directories that are automatically removed if the nightly script
1052 # fails to start correctly
1053 function newdir {
1054     dir=$1
1055     toadd=
1056     while [ ! -d $dir ]; do
1057         toadd="$dir $toadd"
1058         dir='dirname $dir'
1059     done
1060     toadd=
1061     newlist=

```

```

1062     for dir in $toadd; do
1063         if staffer mkdir $dir; then
1064             newlist="$ISUSER $dir $newlist"
1065             toadd="$dir $toadd"
1066         else
1067             [ -z "$storm" ] || staffer rmdir $storm
1068             return 1
1069         fi
1070     done
1071     newdirlist="$newlist $newdirlist"
1072     return 0
1073 }
1074 newdirlist=

1076 [ -d $CODEMGR_WS ] || newdir $CODEMGR_WS || exit 1

1078 # since this script assumes the build is from full source, it nullifies
1079 # variables likely to have been set by a "ws" script; nullification
1080 # confines the search space for headers and libraries to the proto area
1081 # built from this immediate source.
1082 ENVLDLIBS1=
1083 ENVLDLIBS2=
1084 ENVLDLIBS3=
1085 ENVCPPFLAGS1=
1086 ENVCPPFLAGS2=
1087 ENVCPPFLAGS3=
1088 ENVCPPFLAGS4=
1089 PARENT_ROOT=

1091 export ENVLDLIBS3 ENVCPPFLAGS1 ENVCPPFLAGS2 ENVCPPFLAGS3 ENVCPPFLAGS4 \
1092     ENVLDLIBS1 ENVLDLIBS2 PARENT_ROOT

1094 PKGARCHIVE_ORIG=$PKGARCHIVE
1347 IA32_IHV_PKGS_ORIG=$IA32_IHV_PKGS

1096 #
1097 # Juggle the logs and optionally send mail on completion.
1098 #

1100 function logshuffle {
1101     LLOG="$ATLOG/log.`date +%F.%H:%M`"
1102     if [ -f $LLOG -o -d $LLOG ]; then
1103         LLOG=$LLOG.$$
1104     fi
1105     mkdir $LLOG
1106     export LLOG

1108     if [ "$build_ok" = "y" ]; then
1109         mv $ATLOG/proto_list_${MACH} $LLOG

1111         if [ -f $ATLOG/proto_list_tools_${MACH} ]; then
1112             mv $ATLOG/proto_list_tools_${MACH} $LLOG
1113         fi

1115         if [ -f $TMPDIR/wsdiff.results ]; then
1116             mv $TMPDIR/wsdiff.results $LLOG
1117         fi

1119         if [ -f $TMPDIR/wsdiff-nd.results ]; then
1120             mv $TMPDIR/wsdiff-nd.results $LLOG
1121         fi
1122     fi

1124     #
1125     # Now that we're about to send mail, it's time to check the noise
1126     # file. In the event that an error occurs beyond this point, it will

```

```

1127 # be recorded in the nightly.log file, but nowhere else. This would
1128 # include only errors that cause the copying of the noise log to fail
1129 # or the mail itself not to be sent.
1130 #
1131
1132 exec >>$LOGFILE 2>&1
1133 if [ -s $build_noise_file ]; then
1134     echo "\n=== Nightly build noise ===\n" |
1135         tee -a $LOGFILE >>$mail_msg_file
1136     cat $build_noise_file >>$LOGFILE
1137     cat $build_noise_file >>$mail_msg_file
1138     echo | tee -a $LOGFILE >>$mail_msg_file
1139 fi
1140 rm -f $build_noise_file
1141
1142 case "$build_ok" in
1143     y)
1144         state=Completed
1145         ;;
1146     i)
1147         state=Interrupted
1148         ;;
1149     *)
1150         state=Failed
1151         ;;
1152 esac
1153 NIGHTLY_STATUS=$state
1154 export NIGHTLY_STATUS
1155
1156 run_hook POST_NIGHTLY $state
1157 run_hook SYS_POST_NIGHTLY $state
1158
1159 #
1160 # mailx(1) sets From: based on the -r flag
1161 # if it is given.
1162 #
1163 mailx_r=
1164 if [[ -n "${MAILFROM}" ]]; then
1165     mailx_r="-r ${MAILFROM}"
1166 fi
1167
1168 cat $build_time_file $build_envron_file $mail_msg_file \
1169     > ${LLOG}/mail_msg
1170 if [ "$m_FLAG" = "y" ]; then
1171     cat ${LLOG}/mail_msg | /usr/bin/mailx ${mailx_r} -s \
1172     "Nightly ${MACH} Build of 'basename ${CODEMGR_WS}' ${state}." \
1173     ${MAILTO}
1174 fi
1175
1176 if [ "$u_FLAG" = "y" -a "$build_ok" = "y" ]; then
1177     staffer cp ${LLOG}/mail_msg $PARENT_WS/usr/src/mail_msg-${MACH}
1178     staffer cp $LOGFILE $PARENT_WS/usr/src/nightly-${MACH}.log
1179 fi
1180
1181 mv $LOGFILE $LLOG
1182 }

```

unchanged_portion_omitted

```

1259 # Ensure no other instance of this script is running on this host.
1260 # LOCKNAME can be set in <env_file>, and is by default, but is not
1261 # required due to the use of $ATLOG below.
1262 if [ -n "$LOCKNAME" ]; then
1263     create_lock /tmp/$LOCKNAME "lockfile"
1264 fi
1265 #
1266 # Create from one, two, or three other locks:

```

```

1267 # $ATLOG/nightly.lock
1268 # - protects against multiple builds in same workspace
1269 # $PARENT_WS/usr/src/nightly.$MACH.lock
1270 # - protects against multiple 'u' copy-backs
1271 # $NIGHTLY_PARENT_ROOT/nightly.lock
1272 # - protects against multiple 'U' copy-backs
1273 #
1274 # Overriding ISUSER to 1 causes the lock to be created as root if the
1275 # script is run as root. The default is to create it as $STAFFER.
1276 ISUSER=1 create_lock $ATLOG/nightly.lock "atloglockfile"
1277 if [ "$u_FLAG" = "y" ]; then
1278     create_lock $PARENT_WS/usr/src/nightly.$MACH.lock "unlockfile"
1279 fi
1280 if [ "$U_FLAG" = "y" ]; then
1281     # NIGHTLY_PARENT_ROOT is written as root if script invoked as root.
1282     ISUSER=1 create_lock $NIGHTLY_PARENT_ROOT/nightly.lock "Ulockfile"
1283 fi
1284
1285 # Locks have been taken, so we're doing a build and we're committed to
1286 # the directories we may have created so far.
1287 newdirlist=
1288
1289 #
1290 # Create mail_msg_file
1291 #
1292 mail_msg_file="${TMPDIR}/mail_msg"
1293 touch $mail_msg_file
1294 build_time_file="${TMPDIR}/build_time"
1295 build_envron_file="${TMPDIR}/build_envron"
1296 touch $build_envron_file
1297 #
1298 # Move old LOGFILE aside
1299 # ATLOG directory already made by 'create_lock' above
1300 #
1301 if [ -f $LOGFILE ]; then
1302     mv -f $LOGFILE ${LOGFILE}-
1303 fi
1304 #
1305 # Build OsNet source
1306 #
1307 START_DATE='date'
1308 SECONDS=0
1309 echo "\n=== Nightly $maketype build started: $START_DATE ===" \
1310     | tee -a $LOGFILE > $build_time_file
1311
1312 echo "\nBuild project: $build_project\nBuild taskid: $build_taskid" | \
1313     tee -a $mail_msg_file >> $LOGFILE
1314
1315 # make sure we log only to the nightly build file
1316 build_noise_file="${TMPDIR}/build_noise"
1317 exec </dev/null >$build_noise_file 2>&1
1318
1319 run_hook SYS_PRE_NIGHTLY
1320 run_hook PRE_NIGHTLY
1321
1322 echo "\n=== list of environment variables ===\n" >> $LOGFILE
1323 env >> $LOGFILE
1324
1325 echo "\n=== Nightly argument issues ===\n" | tee -a $mail_msg_file >> $LOGFILE
1326
1327 if [ "$P_FLAG" = "y" ]; then
1328     obsolete_build GPROF | tee -a $mail_msg_file >> $LOGFILE
1329 fi
1330
1331 if [ "$T_FLAG" = "y" ]; then
1332     obsolete_build TRACE | tee -a $mail_msg_file >> $LOGFILE
1333 fi

```

```

1586 fi

1327 if [ "$N_FLAG" = "y" ]; then
1328     if [ "$p_FLAG" = "y" ]; then
1329         cat <<EOF | tee -a $mail_msg_file >> $LOGFILE
1330 WARNING: the p option (create packages) is set, but so is the N option (do
1331     not run protocmp); this is dangerous; you should unset the N option
1332 EOF
1333     else
1334         cat <<EOF | tee -a $mail_msg_file >> $LOGFILE
1335 Warning: the N option (do not run protocmp) is set; it probably shouldn't be
1336 EOF
1337     fi
1338     echo "" | tee -a $mail_msg_file >> $LOGFILE
1339 fi

1341 if [ "$D_FLAG" = "n" -a "$l_FLAG" = "y" ]; then
1342     #
1343     # In the past we just complained but went ahead with the lint
1344     # pass, even though the proto area was built non-DEBUG. It's
1345     # unlikely that non-DEBUG headers will make a difference, but
1346     # rather than assuming it's a safe combination, force the user
1347     # to specify a DEBUG build.
1348     #
1349     echo "WARNING: DEBUG build not requested; disabling lint.\n" \
1350         | tee -a $mail_msg_file >> $LOGFILE
1351     l_FLAG=n
1352 fi

1354 if [ "$f_FLAG" = "y" ]; then
1355     if [ "$i_FLAG" = "y" ]; then
1356         echo "WARNING: the -f flag cannot be used during incremental" \
1357             "builds; ignoring -f\n" | tee -a $mail_msg_file >> $LOGFILE
1358         f_FLAG=n
1359     fi
1360     if [ "${l_FLAG}${p_FLAG}" != "yy" ]; then
1361         echo "WARNING: the -f flag requires -l, and -p;" \
1362             "ignoring -f\n" | tee -a $mail_msg_file >> $LOGFILE
1363         f_FLAG=n
1364     fi
1365 fi

1367 if [ "$w_FLAG" = "y" -a ! -d $ROOT ]; then
1368     echo "WARNING: -w specified, but $ROOT does not exist;" \
1369         "ignoring -w\n" | tee -a $mail_msg_file >> $LOGFILE
1370     w_FLAG=n
1371 fi

1373 if [ "$t_FLAG" = "n" ]; then
1374     #
1375     # We're not doing a tools build, so make sure elfsign(1) is
1376     # new enough to safely sign non-crypto binaries. We test
1377     # debugging output from elfsign to detect the old version.
1378     #
1379     newelfsigntest='SUNW_CRYPTO_DEBUG=stderr /usr/bin/elfsign verify \
1380     -e /usr/lib/security/pkcs11_softtoken.so.1 2>&l \
1381     | egrep algorithmOID'
1382     if [ -z "$newelfsigntest" ]; then
1383         echo "WARNING: /usr/bin/elfsign out of date;" \
1384             "will only sign crypto modules\n" | \
1385             tee -a $mail_msg_file >> $LOGFILE
1386         export ELFSIGN_OBJECT=true
1387     elif [ "$VERIFY_ELFSIGN" = "y" ]; then
1388         echo "WARNING: VERIFY_ELFSIGN=y requires" \
1389             "the -t flag; ignoring VERIFY_ELFSIGN\n" | \
1390             tee -a $mail_msg_file >> $LOGFILE

```

```

1391     fi
1392 fi

1394 case $MULTI_PROTO in
1395 yes|no) ;;
1396 *)
1397     echo "WARNING: MULTI_PROTO is \"\$MULTI_PROTO\"; " \
1398         "should be \"yes\" or \"no\"." | tee -a $mail_msg_file >> $LOGFILE
1399     echo "Setting MULTI_PROTO to \"no\".\n" | \
1400         tee -a $mail_msg_file >> $LOGFILE
1401     export MULTI_PROTO=no
1402     ;;
1403 esac

1405 echo "\n=== Build version ===\n" | tee -a $mail_msg_file >> $LOGFILE
1406 echo $VERSION | tee -a $mail_msg_file >> $LOGFILE

1408 # Save the current proto area if we're comparing against the last build
1409 if [ "$w_FLAG" = "y" -a -d "$ROOT" ]; then
1410     if [ -d "$ROOT.prev" ]; then
1411         rm -rf $ROOT.prev
1412     fi
1413     mv $ROOT $ROOT.prev
1414 fi

1416 # Same for non-DEBUG proto area
1417 if [ "$w_FLAG" = "y" -a "$MULTI_PROTO" = yes -a -d "$ROOT-nd" ]; then
1418     if [ -d "$ROOT-nd.prev" ]; then
1419         rm -rf $ROOT-nd.prev
1420     fi
1421     mv $ROOT-nd $ROOT-nd.prev
1422 fi

1424 #
1425 # Echo the SCM type of the parent workspace, this can't just be which_scm
1426 # as that does not know how to identify various network repositories.
1427 #
1428 function parent_wstype {
1429     typeset scm_type junk

1431     CODEMGR_WS="$BRINGOVER_WS" "$WHICH_SCM" 2>/dev/null \
1432     | read scm_type junk
1433     if [[ -z "$scm_type" || "$scm_type" == unknown ]]; then
1434         # Probe BRINGOVER_WS to determine its type
1435         if [[ $BRINGOVER_WS == ssh://* ]]; then
1436             scm_type="subversion"
1437         elif [[ $BRINGOVER_WS == file://* ]] &&
1438             egrep -s "This is a Subversion repository" \
1439                 ${BRINGOVER_WS#file://}/README.txt 2> /dev/null; then
1440             scm_type="subversion"
1441         elif [[ $BRINGOVER_WS == http://* ]] &&
1442             wget -q -O- --save-headers "$BRINGOVER_WS/?cmd=heads" | \
1443             egrep -s "application/mercurial" 2> /dev/null; then
1444             scm_type="mercurial"
1445         elif svn info $BRINGOVER_WS > /dev/null 2>&l; then
1446             scm_type="subversion"
1447         else
1448             scm_type="none"
1449         fi
1450     fi

1452 # fold both unsupported and unrecognized results into "none"
1453 case "$scm_type" in

```

```

1448     mercurial)
1449     none/subversion/teamware/mercurial)
1450 *)
1451     ;;
1452 esac

1454     echo $scm_type
1455 }

1457 # Echo the SCM types of $CODEMGR_WS and $BRINGOVER_WS
1458 function child_wstype {
1459     typeset scm_type junk

1461     # Probe CODEMGR_WS to determine its type
1462     if [[ -d $CODEMGR_WS ]]; then
1463         $WHICH_SCM | read scm_type junk || exit 1
1464     fi

1466     case "$scm_type" in
1467     none|git|mercurial)
1473     none/subversion/git/teamware/mercurial)
1474     ;;
1469     *)
1470     scm_type=none
1471     ;;
1472 esac

1473     echo $scm_type
1474 }

1476 SCM_TYPE=$(child_wstype)

1478 #
1479 # Decide whether to clobber
1480 #
1481 if [ "$i_FLAG" = "n" -a -d "$SRC" ]; then
1482     echo "\n==== Make clobber at 'date' ==== \n" >> $LOGFILE

1484     cd $SRC
1485     # remove old clobber file
1486     rm -f $SRC/clobber.out
1487     rm -f $SRC/clobber-${MACH}.out

1489     # Remove all .make.state* files, just in case we are restarting
1490     # the build after having interrupted a previous 'make clobber'.
1491     find . \( -name SCCS -o -name .hg -o -name .svn -o -name .git \
1492         -o -name 'interfaces.*' \) -prune \
1493         -o -name '*.make.*' -print | xargs rm -f

1495     $MAKE -ek clobber 2>&1 | tee -a $SRC/clobber-${MACH}.out >> $LOGFILE
1496     echo "\n==== Make clobber ERRORS ==== \n" >> $mail_msg_file
1497     grep "$MAKE:" $SRC/clobber-${MACH}.out |
1498     egrep -v "Ignoring unknown host" \
1499     >> $mail_msg_file

1501     if [[ "$t_FLAG" = "y" ]]; then
1502         echo "\n==== Make tools clobber at 'date' ==== \n" >> $LOGFILE
1503         cd ${TOOLS}
1504         rm -f ${TOOLS}/clobber-${MACH}.out
1505         $MAKE TOOLS_PROTO=${TOOLS_PROTO} -ek clobber 2>&1 | \
1506         tee -a ${TOOLS}/clobber-${MACH}.out >> $LOGFILE
1507         echo "\n==== Make tools clobber ERRORS ==== \n" \
1508         >> $mail_msg_file
1509         grep "$MAKE:" ${TOOLS}/clobber-${MACH}.out \
1510         >> $mail_msg_file
1511         rm -rf ${TOOLS_PROTO}

```

```

1512         mkdir -p ${TOOLS_PROTO}
1513     fi

1515     typeset roots=$(allprotos)
1516     echo "\n\nClearing $roots" >> "$LOGFILE"
1517     rm -rf $roots

1519     # Get back to a clean workspace as much as possible to catch
1520     # problems that only occur on fresh workspaces.
1521     # Remove all .make.state* files, libraries, and .o's that may
1522     # have been omitted from clobber. A couple of libraries are
1523     # under source code control, so leave them alone.
1524     # We should probably blow away temporary directories too.
1525     cd $SRC
1526     find $reldirdirs \( -name SCCS -o -name .hg -o -name .svn \
1527         -o -name .git -o -name 'interfaces.*' \) -prune -o \
1528         \( -name '*.make.*' -o -name 'lib*.a' -o -name 'lib*.so*' -o \
1529         -name '*.o' \) -print | \
1530     grep -v 'tools/ctf/dwarf/.*/libdwarf' | xargs rm -f
1531 else
1532     echo "\n==== No clobber at 'date' ==== \n" >> $LOGFILE
1533 fi

1804 type bringover_teamware > /dev/null 2>&1 || function bringover_teamware {
1805     # sleep on the parent workspace's lock
1806     while egrep -s write $BRINGOVER_WS/Codemgr_wsdata/locks
1807     do
1808         sleep 120
1809     done

1811     if [[ -z $BRINGOVER ]]; then
1812         BRINGOVER=$TEAMWARE/bin/bringover
1813     fi

1815     staffer $BRINGOVER -c "nightly update" -p $BRINGOVER_WS \
1816     -w $CODEMGR_WS $BRINGOVER_FILES < /dev/null 2>&1 ||
1817     touch $TMPDIR/bringover_failed

1819     staffer bringovercheck $CODEMGR_WS >$TMPDIR/bringovercheck.out 2>&1
1820     if [ -s $TMPDIR/bringovercheck.out ]; then
1821         echo "\n==== POST-BRINGOVER CLEANUP NOISE ==== \n"
1822         cat $TMPDIR/bringovercheck.out
1823     fi
1824 }

1535 type bringover_mercurial > /dev/null 2>&1 || function bringover_mercurial {
1536     typeset -x PATH=$PATH

1538     # If the repository doesn't exist yet, then we want to populate it.
1539     if [[ ! -d $CODEMGR_WS/.hg ]]; then
1540         staffer hg init $CODEMGR_WS
1541         staffer echo "[paths]" > $CODEMGR_WS/.hg/hgrc
1542         staffer echo "default=$BRINGOVER_WS" >> $CODEMGR_WS/.hg/hgrc
1543         touch $TMPDIR/new_repository
1544     fi

1546     typeset -x HGMERGE="/bin/false"

1548     #
1549     # If the user has changes, regardless of whether those changes are
1550     # committed, and regardless of whether those changes conflict, then
1551     # we'll attempt to merge them either implicitly (uncommitted) or
1552     # explicitly (committed).
1553     #
1554     # These are the messages we'll use to help clarify mercurial output
1555     # in those cases.

```

```

1556 #
1557 typeset mergefailmsg="\
1558 ***\n\
1559 *** nightly was unable to automatically merge your changes. You should\n\
1560 *** redo the full merge manually, following the steps outlined by mercurial\n\
1561 *** above, then restart nightly.\n\
1562 ***\n"
1563 typeset mergepassmsg="\
1564 ***\n\
1565 *** nightly successfully merged your changes. This means that your working\n\
1566 *** directory has been updated, but those changes are not yet committed.\n\
1567 *** After nightly completes, you should validate the results of the merge,\n\
1568 *** then use hg commit manually.\n\
1569 ***\n"

1571 #
1572 # For each repository in turn:
1573 #
1574 # 1. Do the pull. If this fails, dump the output and bail out.
1575 #
1576 # 2. If the pull resulted in an extra head, do an explicit merge.
1577 # If this fails, dump the output and bail out.
1578 #
1579 # Because we can't rely on Mercurial to exit with a failure code
1580 # when a merge fails (Mercurial issue #186), we must grep the
1581 # output of pull/merge to check for attempted and/or failed merges.
1582 #
1583 # 3. If a merge failed, set the message and fail the bringover.
1584 #
1585 # 4. Otherwise, if a merge succeeded, set the message
1586 #
1587 # 5. Dump the output, and any message from step 3 or 4.
1588 #

1590 typeset HG_SOURCE=$BRINGOVER_WS
1591 if [ ! -f $TMPDIR/new_repository ]; then
1592     HG_SOURCE=$TMPDIR/open_bundle.hg
1593     staffer hg --cwd $CODEMGR_WS incoming --bundle $HG_SOURCE \
1594         -v $BRINGOVER_WS > $TMPDIR/incoming_open.out

1596 #
1597 # If there are no incoming changesets, then incoming will
1598 # fail, and there will be no bundle file. Reset the source,
1599 # to allow the remaining logic to complete with no false
1600 # negatives. (Unlike incoming, pull will return success
1601 # for the no-change case.)
1602 #
1603 # if (( $? != 0 )); then
1604     HG_SOURCE=$BRINGOVER_WS
1605 fi
1606 fi

1608 staffer hg --cwd $CODEMGR_WS pull -u $HG_SOURCE \
1609     > $TMPDIR/pull_open.out 2>&1
1610 if (( $? != 0 )); then
1611     printf "%s: pull failed as follows:\n\n" "$CODEMGR_WS"
1612     cat $TMPDIR/pull_open.out
1613     if grep "^merging.*failed" $TMPDIR/pull_open.out > /dev/null 2>&1
1614     then
1615         printf "$mergefailmsg"
1616     fi
1617     touch $TMPDIR/bringover_failed
1618     return
1619 fi

1620 if grep "not updating" $TMPDIR/pull_open.out > /dev/null 2>&1; then
1621     staffer hg --cwd $CODEMGR_WS merge \

```

```

1622     >> $TMPDIR/pull_open.out 2>&1
1623     if (( $? != 0 )); then
1624         printf "%s: merge failed as follows:\n\n" \
1625             "$CODEMGR_WS"
1626         cat $TMPDIR/pull_open.out
1627         if grep "^merging.*failed" $TMPDIR/pull_open.out \
1628             > /dev/null 2>&1; then
1629             printf "$mergefailmsg"
1630         fi
1631         touch $TMPDIR/bringover_failed
1632         return
1633     fi
1634 fi

1636 printf "updated %s with the following results:\n" "$CODEMGR_WS"
1637 cat $TMPDIR/pull_open.out
1638 if grep "^merging" $TMPDIR/pull_open.out > /dev/null 2>&1; then
1639     printf "$mergepassmsg"
1640 fi
1641 printf "\n"

1643 #
1644 # Per-changeset output is neither useful nor manageable for a
1645 # newly-created repository.
1646 #
1647 if [ -f $TMPDIR/new_repository ]; then
1648     return
1649 fi

1651 printf "\nadded the following changesets to open repository:\n"
1652 cat $TMPDIR/incoming_open.out

1645 #
1646 # The closed repository could have been newly created, even though
1647 # the open one previously existed...
1648 #
1649 if [ -f $TMPDIR/new_closed ]; then
1650     return
1651 fi

1653 if [ -f $TMPDIR/incoming_closed.out ]; then
1654     printf "\nadded the following changesets to closed repository:\n"
1655     cat $TMPDIR/incoming_closed.out
1656 fi
1657 }

1659 type bringover_subversion > /dev/null 2>&1 || function bringover_subversion {
1660     typeset -x PATH=$PATH

1662     if [[ ! -d $CODEMGR_WS/.svn ]]; then
1663         staffer svn checkout $BRINGOVER_WS $CODEMGR_WS ||
1664             touch $TMPDIR/bringover_failed
1665     else
1666         typeset root
1667         root=$(staffer svn info $CODEMGR_WS |
1668             awk '/^Repository Root:/ {print $NF}')
1669         if [[ $root != $BRINGOVER_WS ]]; then
1670             # We fail here because there's no way to update
1671             # from a named repo.
1672             cat <<-EOF
1673             \ $BRINGOVER_WS doesn't match repository root:
1674             \ $BRINGOVER_WS: $BRINGOVER_WS
1675             Repository root: $root
1676             EOF
1677             touch $TMPDIR/bringover_failed
1678         else

```



```

1979 # If a conflict happens, svn still exits 0.
1980 staffer svn update $CODEMGR_WS | tee $TMPDIR/pull.out ||
1981 touch $TMPDIR/bringover_failed
1982 if grep "^C" $TMPDIR/pull.out > /dev/null 2>&1; then
1983 touch $TMPDIR/bringover_failed
1984 fi
1985 fi
1986 }
1653 }
_____unchanged_portion_omitted_____

1660 #
1661 # Decide whether to bringover to the codemgr workspace
1662 #
1663 if [ "$_FLAG" = "n" ]; then
1664 PARENT_SCM_TYPE=$(parent_wstype)
1666 if [[ $SCM_TYPE != none && $SCM_TYPE != $PARENT_SCM_TYPE ]]; then
1667 echo "cannot bringover from $PARENT_SCM_TYPE to $SCM_TYPE, " \
1668 "quitting at 'date'." | tee -a $mail_msg_file >> $LOGFILE
1669 exit 1
1670 fi
1672 run_hook PRE_BRINGOVER
1674 echo "\n==== bringover to $CODEMGR_WS at 'date' ==== \n" >> $LOGFILE
1675 echo "\n==== BRINGOVER LOG ==== \n" >> $mail_msg_file
1677 eval "bringover_${PARENT_SCM_TYPE}" 2>&1 |
1678 tee -a $mail_msg_file >> $LOGFILE
1680 if [ -f $TMPDIR/bringover_failed ]; then
1681 rm -f $TMPDIR/bringover_failed
1682 build_ok=n
1683 echo "trouble with bringover, quitting at 'date'." |
1684 tee -a $mail_msg_file >> $LOGFILE
1685 exit 1
1686 fi
1688 #
1689 # It's possible that we used the bringover above to create
1690 # $CODEMGR_WS. If so, then SCM_TYPE was previously "none,"
1691 # but should now be the same as $BRINGOVER_WS.
1692 #
1693 [[ $SCM_TYPE = none ]] && SCM_TYPE=$PARENT_SCM_TYPE
1695 run_hook POST_BRINGOVER
1697 check_closed_bins
2031 check_closed_tree
1699 else
1700 echo "\n==== No bringover to $CODEMGR_WS ==== \n" >> $LOGFILE
1701 fi
1703 # Safeguards
1704 [[ -v CODEMGR_WS ]] || fatal_error "Error: Variable CODEMGR_WS not set."
1705 [[ -d "${CODEMGR_WS}" ]] || fatal_error "Error: ${CODEMGR_WS} is not a directory
1706 [[ -f "${CODEMGR_WS}/usr/src/Makefile" ]] || fatal_error "Error: ${CODEMGR_WS}/u
1708 echo "\n==== Build environment ==== \n" | tee -a $build_envIRON_file >> $LOGFILE
1710 # System
1711 whence uname | tee -a $build_envIRON_file >> $LOGFILE
1712 uname -a 2>&1 | tee -a $build_envIRON_file >> $LOGFILE
1713 echo | tee -a $build_envIRON_file >> $LOGFILE

```

```

1715 # make
1716 whence $MAKE | tee -a $build_envIRON_file >> $LOGFILE
1717 $MAKE -v | tee -a $build_envIRON_file >> $LOGFILE
1718 echo "number of concurrent jobs = $MAKE_MAX_JOBS" |
1719 tee -a $build_envIRON_file >> $LOGFILE
1721 #
1722 # Report the compiler versions.
1723 #
1725 if [[ ! -f $SRC/Makefile ]]; then
1726 build_ok=n
1727 echo "\nUnable to find \"Makefile\" in $SRC." | \
1728 tee -a $build_envIRON_file >> $LOGFILE
1729 exit 1
1730 fi
1732 ( cd $SRC
1733 for target in cc-version cc64-version java-version; do
1734 echo
1735 #
1736 # Put statefile somewhere we know we can write to rather than trip
1737 # over a read-only $srcroot.
1738 #
1739 rm -f $TMPDIR/make-state
1740 export SRC
1741 if $MAKE -K $TMPDIR/make-state -e $target 2>/dev/null; then
1742 continue
1743 fi
1744 touch $TMPDIR/nocompiler
1745 done
1746 echo
1747 ) | tee -a $build_envIRON_file >> $LOGFILE
1749 if [ -f $TMPDIR/nocompiler ]; then
1750 rm -f $TMPDIR/nocompiler
1751 build_ok=n
1752 echo "Aborting due to missing compiler." |
1753 tee -a $build_envIRON_file >> $LOGFILE
1754 exit 1
1755 fi
1757 # as
1758 whence as | tee -a $build_envIRON_file >> $LOGFILE
1759 as -V 2>&1 | head -1 | tee -a $build_envIRON_file >> $LOGFILE
1760 echo | tee -a $build_envIRON_file >> $LOGFILE
1762 # Check that we're running a capable link-editor
1763 whence ld | tee -a $build_envIRON_file >> $LOGFILE
1764 LDVER='ld -V 2>&1'
1765 echo $LDVER | tee -a $build_envIRON_file >> $LOGFILE
1766 LDVER='echo $LDVER | sed -e "s/.*-1\\.([0-9]*)\\.*/\\1/'
1767 if [ `expr $LDVER < 422` -eq 1 ]; then
1768 echo "The link-editor needs to be at version 422 or higher to build" | \
1769 tee -a $build_envIRON_file >> $LOGFILE
1770 echo "the latest stuff. Hope your build works." | \
1771 tee -a $build_envIRON_file >> $LOGFILE
1772 fi
1774 #
1775 # Build and use the workspace's tools if requested
1776 #
1777 if [[ "$_FLAG" = "y" ]]; then
1778 set_non_debug_build_flags

```

```

1780     build_tools ${TOOLS_PROTO}
1781     if [[ $? != 0 && "$t_FLAG" = y ]]; then
1782         use_tools $TOOLS_PROTO
1783     fi
1784 fi

2120 #
2121 # copy ihv proto area in addition to the build itself
2122 #
2123 if [ "$X_FLAG" = "y" ]; then
2124     copy_ihv_proto
2125 fi

1786 # timestamp the start of the normal build; the findunref tool uses it.
1787 touch $SRC/.build.tstamp

1789 normal_build

1791 ORIG_SRC=$SRC
1792 BINARCHIVE=${CODEMGR_WS}/bin-${MACH}.cpio.Z

1795 #
1796 # There are several checks that need to look at the proto area, but
1797 # they only need to look at one, and they don't care whether it's
1798 # DEBUG or non-DEBUG.
1799 #
1800 if [[ "$MULTI_PROTO" = yes && "$D_FLAG" = n ]]; then
1801     checkroot=$ROOT-nd
1802 else
1803     checkroot=$ROOT
1804 fi

1806 if [ "$build_ok" = "y" ]; then
1807     echo "\n=== Creating protolist system file at `date` ===" \
1808         >> $LOGFILE
1809     protolist $checkroot > $ATLOG/proto_list_${MACH}
1810     echo "=== protolist system file created at `date` ===\n" \
1811         >> $LOGFILE

1813     if [ "$N_FLAG" != "y" ]; then

1815         E1=
1816         f1=
1817         if [ -d "$SRC/pkgdefs" ]; then
1818             f1="$SRC/pkgdefs/etc/exception_list_${MACH}"
1819             if [ "$X_FLAG" = "y" ]; then
1820                 f1="$f1 $IA32_IHV_WS/usr/src/pkgdefs/etc/excepti
1821             fi
1822         fi

1817     for f in $f1; do
1818         if [ -f "$f" ]; then
1819             E1="$E1 -e $f"
1820         fi
1821     done

1823     E2=
1824     f2=
1825     if [ -d "$SRC/pkg" ]; then
1826         f2="$f2 exceptions/packaging"
1827     fi

1829     for f in $f2; do
1830         if [ -f "$f" ]; then
1831             E2="$E2 -e $f"

```

```

1832     fi
1833 done

2183     if [ -f "$REF_PROTO_LIST" ]; then
2184         #
2185         # For builds that copy the IHV proto area (-X), add the
2186         # IHV proto list to the reference list if the reference
2187         # was built without -X.
2188         #
2189         # For builds that don't copy the IHV proto area, add the
2190         # IHV proto list to the build's proto list if the
2191         # reference was built with -X.
2192         #
2193         # Use the presence of the first file entry of the cached
2194         # IHV proto list in the reference list to determine
2195         # whether it was built with -X or not.
2196         #
2197         IHV_REF_PROTO_LIST=$SRC/pkg/proto_list_ihv_${MACH}
2198         grepfor=$(nawk ' $1 == "f" { print $2; exit }' \
2199             $IHV_REF_PROTO_LIST 2> /dev/null)
2200         if [ $? = 0 -a -n "$grepfor" ]; then
2201             if [ "$X_FLAG" = "y" ]; then
2202                 grep -w "$grepfor" \
2203                     $REF_PROTO_LIST > /dev/null
2204             if [ ! "$?" = "0" ]; then
2205                 REF_IHV_PROTO="-d $IHV_REF_PROTO
2206             fi
2207         else
2208             grep -w "$grepfor" \
2209                 $REF_PROTO_LIST > /dev/null
2210             if [ "$?" = "0" ]; then
2211                 IHV_PROTO_LIST="$IHV_REF_PROTO_L
2212             fi
2213         fi
2214     fi
2215 fi
2216 fi

2218     if [ "$N_FLAG" != "y" -a -f $SRC/pkgdefs/Makefile ]; then
2219         echo "\n=== Impact on SVr4 packages ===\n" >> $mail_msg_file
2220         #
2221         # Compare the build's proto list with current package
2222         # definitions to audit the quality of package
2223         # definitions and makefile install targets. Use the
2224         # current exception list.
2225         #
2226         PKGDEFS_LIST=""
2227         for d in $abssrcdirs; do
2228             if [ -d $d/pkgdefs ]; then
2229                 PKGDEFS_LIST="$PKGDEFS_LIST -d $d/pkgdefs"
2230             fi
2231         done
2232         if [ "$X_FLAG" = "y" -a \
2233             -d $IA32_IHV_WS/usr/src/pkgdefs ]; then
2234             PKGDEFS_LIST="$PKGDEFS_LIST -d $IA32_IHV_WS/usr/src/pkgd
2235         fi
2236         $PROTOCMPTRSE \
2237             "Files missing from the proto area:" \
2238             "Files missing from packages:" \
2239             "Inconsistencies between pkgdefs and proto area:" \
2240             ${E1} \
2241             ${PKGDEFS_LIST} \
2242             $ATLOG/proto_list_${MACH} \
2243             >> $mail_msg_file
2234     fi

```

```

1836     if [ "$N_FLAG" != "y" -a -d $SRC/pkg ]; then
1837         echo "\n==== Validating manifests against proto area ==== \n" \
1838             >> $mail_msg_file
1839         ( cd $SRC/pkg ; $MAKE -e protocmp ROOT="$checkroot" ) \
1840             >> $mail_msg_file
1842     fi
1844     if [ "$N_FLAG" != "y" -a -f "$REF_PROTO_LIST" ]; then
1845         echo "\n==== Impact on proto area ==== \n" >> $mail_msg_file
1846         if [ -n "$E2" ]; then
1847             ELIST=$E2
1848         else
1849             ELIST=$E1
1850         fi
1851         $PROTOCMPTERSE \
1852             "Files in yesterday's proto area, but not today's:" \
1853             "Files in today's proto area, but not yesterday's:" \
1854             "Files that changed between yesterday and today:" \
1855             ${ELIST} \
1856             -d $REF_PROTO_LIST \
2267             $REF_IHV_PROTO \
1857             $ATLOG/proto_list_{$MACH} \
2269             $IHV_PROTO_LIST \
1858             >> $mail_msg_file
1859     fi
1860 fi
1862 if [ "$U_FLAG" = "y" -a "$build_ok" = "y" ]; then
1863     staffer cp $ATLOG/proto_list_{$MACH} \
1864         $PARENT_WS/usr/src/proto_list_{$MACH}
1865 fi
1867 # Update parent proto area if necessary. This is done now
1868 # so that the proto area has either DEBUG or non-DEBUG kernels.
1869 # Note that this clears out the lock file, so we can dispense with
1870 # the variable now.
1871 if [ "$U_FLAG" = "y" -a "$build_ok" = "y" ]; then
1872     echo "\n==== Copying proto area to $NIGHTLY_PARENT_ROOT ==== \n" | \
1873         tee -a $LOGFILE >> $mail_msg_file
1874     rm -rf $NIGHTLY_PARENT_ROOT/*
1875     unset Ulockfile
1876     mkdir -p $NIGHTLY_PARENT_ROOT
1877     if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
1878         ( cd $ROOT; tar cf - . |
1879             ( cd $NIGHTLY_PARENT_ROOT; umask 0; tar xpf - ) ) 2>&1 |
1880             tee -a $mail_msg_file >> $LOGFILE
1881     fi
1882     if [[ "$MULTI_PROTO" = yes && "$F_FLAG" = n ]]; then
1883         rm -rf $NIGHTLY_PARENT_ROOT-nd/*
1884         mkdir -p $NIGHTLY_PARENT_ROOT-nd
1885         cd $ROOT-nd
1886         ( tar cf - . |
1887             ( cd $NIGHTLY_PARENT_ROOT-nd; umask 0; tar xpf - ) ) 2>&1 |
1888             tee -a $mail_msg_file >> $LOGFILE
1889     fi
1890     if [ -n "${NIGHTLY_PARENT_TOOLS_ROOT}" ]; then
1891         echo "\n==== Copying tools proto area to $NIGHTLY_PARENT_TOOLS_ROOT"
1892             >> $mail_msg_file
1893         rm -rf $NIGHTLY_PARENT_TOOLS_ROOT/*
1894         mkdir -p $NIGHTLY_PARENT_TOOLS_ROOT
1895         if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
1896             ( cd $TOOLS_PROTO; tar cf - . |
1897                 ( cd $NIGHTLY_PARENT_TOOLS_ROOT;
1898                     umask 0; tar xpf - ) ) 2>&1 |
1899                 tee -a $mail_msg_file >> $LOGFILE

```

```

1900         fi
1901     fi
1902 fi
1904 #
1905 # ELF verification: ABI (-A) and runtime (-r) checks
1906 #
1907 if [[ ($build_ok = y) && ( ($A_FLAG = y) || ($r_FLAG = y) ) ]]; then
1908     # Directory ELF-data.$MACH holds the files produced by these tests.
1909     elf_ddir=$SRC/ELF-data.$MACH
1911     # If there is a previous ELF-data backup directory, remove it. Then,
1912     # rotate current ELF-data directory into its place and create a new
1913     # empty directory
1914     rm -rf $elf_ddir.ref
1915     if [[ -d $elf_ddir ]]; then
1916         mv $elf_ddir $elf_ddir.ref
1917     fi
1918     mkdir -p $elf_ddir
1920     # Call find_elf to produce a list of the ELF objects in the proto area.
1921     # This list is passed to check_rtime and interface_check, preventing
1922     # them from separately calling find_elf to do the same work twice.
1923     find_elf -fr $checkroot > $elf_ddir/object_list
1925     if [[ $A_FLAG = y ]]; then
1926         echo "\n==== Check versioning and ABI information ==== \n" | \
1927             tee -a $LOGFILE >> $mail_msg_file
1929     # Produce interface description for the proto. Report errors.
1930     interface_check -o -w $elf_ddir -f object_list \
1931         -i interface -E interface.err
1932     if [[ -s $elf_ddir/interface.err ]]; then
1933         tee -a $LOGFILE < $elf_ddir/interface.err \
1934             >> $mail_msg_file
1935     fi
1937     # If ELF_DATA_BASELINE_DIR is defined, compare the new interface
1938     # description file to that from the baseline gate. Issue a
1939     # warning if the baseline is not present, and keep going.
1940     if [[ "$ELF_DATA_BASELINE_DIR" != '' ]]; then
1941         base_ifile="$ELF_DATA_BASELINE_DIR/interface"
1943         echo "\n==== Compare versioning and ABI information" \
1944             "to baseline ==== \n" | \
1945             tee -a $LOGFILE >> $mail_msg_file
1946         echo "Baseline: $base_ifile\n" >> $LOGFILE
1948     fi
1949     if [[ -f $base_ifile ]]; then
1950         interface_cmp -d -o $base_ifile \
1951             $elf_ddir/interface > $elf_ddir/interface.cm
1952         if [[ -s $elf_ddir/interface.cm ]]; then
1953             echo | tee -a $LOGFILE >> $mail_msg_file
1954             tee -a $LOGFILE < \
1955                 $elf_ddir/interface.cm \
1956                 >> $mail_msg_file
1957         else
1958             echo "baseline not available. comparison" \
1959                 "skipped" | \
1960                 tee -a $LOGFILE >> $mail_msg_file
1961     fi
1963 fi
1964 fi

```

```

1966     if [[ $r_FLAG = y ]]; then
1967         echo "\n==== Check ELF runtime attributes ==== \n" | \
1968             tee -a $LOGFILE >> $mail_msg_file

1970     # If we're doing a DEBUG build the proto area will be left
1971     # with debuggable objects, thus don't assert -s.
1972     if [[ $D_FLAG = y ]]; then
1973         rtime_sflag=""
1974     else
1975         rtime_sflag="-s"
1976     fi
1977     check_rtime -i -m -v $rtime_sflag -o -w $elf_ddir \
1978         -D object_list -f object_list -E runtime.err \
1979         -I runtime.attr.raw

1981     # check_rtime -I output needs to be sorted in order to
1982     # compare it to that from previous builds.
1983     sort $elf_ddir/runtime.attr.raw > $elf_ddir/runtime.attr
1984     rm $elf_ddir/runtime.attr.raw

1986     # Report errors
1987     if [[ -s $elf_ddir/runtime.err ]]; then
1988         tee -a $LOGFILE < $elf_ddir/runtime.err \
1989             >> $mail_msg_file
1990     fi

1992     # If there is an ELF-data directory from a previous build,
1993     # then diff the attr files. These files contain information
1994     # about dependencies, versioning, and runpaths. There is some
1995     # overlap with the ABI checking done above, but this also
1996     # flushes out non-ABI interface differences along with the
1997     # other information.
1998     echo "\n==== Diff ELF runtime attributes" \
1999         "(since last build) ==== \n" | \
2000         tee -a $LOGFILE >> $mail_msg_file >> $mail_msg_file

2002     if [[ -f $elf_ddir.ref/runtime.attr ]]; then
2003         diff $elf_ddir.ref/runtime.attr \
2004             $elf_ddir/runtime.attr \
2005             >> $mail_msg_file
2006     fi
2007 fi

2009 # If -u set, copy contents of ELF-data.$MACH to the parent workspace.
2010 if [[ "$u_FLAG" = "y" ]]; then
2011     p_elf_ddir=$PARENT_WS/usr/src/ELF-data.$MACH

2013     # If parent lacks the ELF-data.$MACH directory, create it
2014     if [[ ! -d $p_elf_ddir ]]; then
2015         staffer mkdir -p $p_elf_ddir
2016     fi

2018     # These files are used asynchronously by other builds for ABI
2019     # verification, as above for the -A option. As such, we require
2020     # the file replacement to be atomic. Copy the data to a temp
2021     # file in the same filesystem and then rename into place.
2022     (
2023         cd $elf_ddir
2024         for elf_dfile in *; do
2025             staffer cp $elf_dfile \
2026                 ${p_elf_ddir}/${elf_dfile}.new
2027             staffer mv -f ${p_elf_ddir}/${elf_dfile}.new \
2028                 ${p_elf_ddir}/${elf_dfile}
2029         done
2030     )
2031 fi

```

```

2032 fi

2034 # DEBUG lint of kernel begins

2036 if [ "$i_CMD_LINE_FLAG" = "n" -a "$l_FLAG" = "y" ]; then
2037     if [ "$LINTDIRS" = "" ]; then
2038         # LINTDIRS="$SRC/uts y $SRC/stand y $SRC/psm y"
2039         LINTDIRS="$SRC y"
2040     fi
2041     set $LINTDIRS
2042     while [ $# -gt 0 ]; do
2043         dolint $1 $2; shift; shift
2044     done
2045 else
2046     echo "\n==== No '$MAKE lint' ==== \n" >> $LOGFILE
2047 fi

2049 # "make check" begins

2051 if [ "$i_CMD_LINE_FLAG" = "n" -a "$C_FLAG" = "y" ]; then
2052     # remove old check.out
2053     rm -f $SRC/check.out

2055     rm -f $SRC/check-${MACH}.out
2056     cd $SRC
2057     $MAKE -ek check ROOT="$checkroot" 2>&1 | tee -a $SRC/check-${MACH}.out \
2058         >> $LOGFILE
2059     echo "\n==== cstyle/hdrchk errors ==== \n" >> $mail_msg_file

2061     grep ":" $SRC/check-${MACH}.out |
2062         egrep -v "Ignoring unknown host" | \
2063         sort | uniq >> $mail_msg_file
2064 else
2065     echo "\n==== No '$MAKE check' ==== \n" >> $LOGFILE
2066 fi

2068 echo "\n==== Find core files ==== \n" | \
2069     tee -a $LOGFILE >> $mail_msg_file

2071 find $abssrkdirs -name core -a -type f -exec file {} \; | \
2072     tee -a $LOGFILE >> $mail_msg_file

2074 if [ "$f_FLAG" = "y" -a "$build_ok" = "y" ]; then
2075     echo "\n==== Diff unreferenced files (since last build) ==== \n" \
2076         | tee -a $LOGFILE >> $mail_msg_file
2077     rm -f $SRC/unref-${MACH}.ref
2078     if [ -f $SRC/unref-${MACH}.out ]; then
2079         mv $SRC/unref-${MACH}.out $SRC/unref-${MACH}.ref
2080     fi

2082     findunref -S $SCM_TYPE -t $SRC/.build.tstamp -s usr $CODEMGR_WS \
2083         ${TOOLS}/findunref/exception_list 2>> $mail_msg_file | \
2084         sort > $SRC/unref-${MACH}.out

2086     if [ ! -f $SRC/unref-${MACH}.ref ]; then
2087         cp $SRC/unref-${MACH}.out $SRC/unref-${MACH}.ref
2088     fi

2090     diff $SRC/unref-${MACH}.ref $SRC/unref-${MACH}.out >> $mail_msg_file
2091 fi

2505 #
2506 # Generate the OpenSolaris deliverables if requested. Some of these
2507 # steps need to come after findunref and are commented below.
2508 #

```

```

2093 # Verify that the usual lists of files, such as exception lists,
2094 # contain only valid references to files.  If the build has failed,
2095 # then don't check the proto area.
2096 CHECK_PATHS=${CHECK_PATHS:-y}
2097 if [ "$CHECK_PATHS" = y -a "$N_FLAG" != y ]; then
2098     echo "\n==== Check lists of files ==== \n" | tee -a $LOGFILE \
2099         >>$mail_msg_file
2100     arg=-b
2101     [ "$build_ok" = y ] && arg=
2102     checkpaths $arg $checkroot 2>&1 | tee -a $LOGFILE >>$mail_msg_file
2103 fi

2105 if [ "$M_FLAG" != "y" -a "$build_ok" = y ]; then
2106     echo "\n==== Impact on file permissions ==== \n" \
2107         >> $mail_msg_file

2109     abspkgdefs=
2110     abspkg=
2111     for d in $absrsrcdirs; do
2112         if [ -d "$d/pkgdefs" ]; then
2113             abspkgdefs="$abspkgdefs $d"
2114         fi
2115     done

2116     if [ -n "$abspkgdefs" ]; then
2117         pmodes -qvdP \
2118             'find $abspkgdefs -name pkginfo.tmpl -print -o \
2119             -name .del\* -prune | sed -e 's:/pkginfo.tmpl$::' | \
2120             sort -u' >> $mail_msg_file
2121     fi

2122     if [ -n "$abspkg" ]; then
2123         for d in "$abspkg"; do
2124             ( cd $d/pkg ; $MAKE -e pmodes ) >> $mail_msg_file
2125         done
2126     fi

2127     if [ "$w_FLAG" = "y" -a "$build_ok" = "y" ]; then
2128         if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
2129             do_wsdiff DEBUG $ROOT.prev $ROOT
2130         fi
2131     fi

2132     if [[ "$MULTI_PROTO" = yes && "$F_FLAG" = n ]]; then
2133         do_wsdiff non-DEBUG $ROOT-nd.prev $ROOT-nd
2134     fi

2135 END_DATE=`date`
2136 echo "==== Nightly $maketype build completed: $END_DATE ==== | \
2137     tee -a $LOGFILE >> $build_time_file

2138 typeset -i10 hours
2139 typeset -Z2 minutes
2140 typeset -Z2 seconds

2141 elapsed_time=$SECONDS
2142 ((hours = elapsed_time / 3600 ))
2143 ((minutes = elapsed_time / 60 % 60))
2144 ((seconds = elapsed_time % 60))

2145 echo "\n==== Total build time ==== | \
2146     tee -a $LOGFILE >> $build_time_file

```

```

2148 echo "\nreal    ${hours}:${minutes}:${seconds}" | \
2149     tee -a $LOGFILE >> $build_time_file

2151 if [ "$u_FLAG" = "y" -a "$f_FLAG" = "y" -a "$build_ok" = "y" ]; then
2152     staffer cp ${SRC}/unref-${MACH}.out $PARENT_WS/usr/src/

2153     #
2154     # Produce a master list of unreferenced files -- ideally, we'd
2155     # generate the master just once after all of the nightlies
2156     # have finished, but there's no simple way to know when that
2157     # will be.  Instead, we assume that we're the last nightly to
2158     # finish and merge all of the unref-${MACH}.out files in
2159     # $PARENT_WS/usr/src/.  If we are in fact the final ${MACH} to
2160     # finish, then this file will be the authoritative master
2161     # list.  Otherwise, another ${MACH}'s nightly will eventually
2162     # overwrite ours with its own master, but in the meantime our
2163     # temporary "master" will be no worse than any older master
2164     # which was already on the parent.
2165     #
2166     set -- $PARENT_WS/usr/src/unref-*.out
2167     cp "$1" ${TMPDIR}/unref.merge
2168     shift

2169     for unref; do
2170         comm -12 ${TMPDIR}/unref.merge "$unref" > ${TMPDIR}/unref.$$
2171         mv ${TMPDIR}/unref.$$ ${TMPDIR}/unref.merge
2172     done

2173     staffer cp ${TMPDIR}/unref.merge $PARENT_WS/usr/src/unrefmaster.out
2174 fi

2175 #
2176 # All done save for the sweeping up.
2177 # (whichever exit we hit here will trigger the "cleanup" trap which
2178 # optionally sends mail on completion).
2179 #
2180 if [ "$build_ok" = "y" ]; then
2181     exit 0
2182 fi
2183 exit 1

```