

```

*****
10083 Tue Mar 11 16:40:35 2014
new/usr/src/tools/scripts/bldenv.sh
4680 nightly and bldenv need to set LC_ALL if they want to fully override the l
*****
_____
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_ALL=C \
159     LANG=C \
160 #endif /* ! codereview */
161     LC_COLLATE=C \
162     LC_CTYPE=C \
163     LC_MESSAGES=C \
164     LC_MONETARY=C \
165     LC_NUMERIC=C \
166     LC_TIME=C

168 # clear environment variables we know to be bad for the build

```

```

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

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

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

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

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

230 # must match the getopt in nightly.sh
231 OPTIND=1
232 NIGHTLY_OPTIONS="-${NIGHTLY_OPTIONS#-}"
233 while getopts '+0ABCDdFfGgIiLmMnNpRrTtUuWw' FLAG "$NIGHTLY_OPTIONS"
234 do

```

```

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

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

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

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

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

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

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

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

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

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

297     export CTFCVTPTBL="${TOOLS_PROTO}/opt/onbld/bin/ctfcvtpdbl"
298     export CTFFINDMOD="${TOOLS_PROTO}/opt/onbld/bin/ctffindmod"

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

```

```

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

305 export DMAKE_MODE=${DMAKE_MODE:-parallel}

307 DEF_STRIPFLAG="--s"

309 TMPDIR="/tmp"

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

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

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

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

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

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

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

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

365 elif "${flags.t}" ; then
366     printf \

```

```
367         'The tools can be (re)built with the install target in %s.\n\n' \  
368         "${TOOLS}"\  
369 fi\  
  
371 #\  
372 # place ourselves in a new task, respecting BUILD_PROJECT if set.  
373 #\  
374 /usr/bin/newtask -c $$ ${BUILD_PROJECT:+-p$BUILD_PROJECT}  
  
376 if [[ "${flags.c}" == "false" && -x "$SHELL" && \  
377     "$(basename -- "${SHELL}")" != "csh" ]]; then  
378     # $SHELL is set, and it's not csh.  
  
380     if "${flags.f}" ; then  
381         print 'WARNING: -f is ignored when $SHELL is not csh'  
382     fi  
  
384     printf 'Using %s as shell.\n' "$SHELL"  
385     exec "$SHELL" ${@:+-c "$@"}  
  
387 elif "${flags.f}" ; then  
388     print 'Using csh -f as shell.'  
389     exec csh -f ${@:+-c "$@"}  
  
391 else  
392     print 'Using csh as shell.'  
393     exec csh ${@:+-c "$@"}  
394 fi  
  
396 # not reached
```

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

1

```
*****
60437 Tue Mar 11 16:40:35 2014
new/usr/src/tools/scripts/nightly.sh
4680 nightly and bldenv need to set LC_ALL if they want to fully override the l
*****
_____unchanged_portion_omitted_____

621 MACH=`uname -p`

623 if [ "$OPTHOME" = "" ]; then
624     OPTHOME=/opt
625     export OPTHOME
626 fi

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

630 Where:
631  -i      Fast incremental options (no clobber, lint, check)
632  -n      Do not do a bringover
633  +t      Use the build tools in $ONBLD_TOOLS/bin
634  -V VERS set the build version string to VERS

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

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

644  -A      check for ABI differences in .so files
645  -C      check for cstyle/hdrchk errors
646  -D      do a build with DEBUG on
647  -F      do _not_ do a non-DEBUG build
648  -G      gate keeper default group of options (-au)
649  -I      integration engineer default group of options (-ampu)
650  -M      do not run pmodes (safe file permission checker)
651  -N      do not run protocmp
652  -R      default group of options for building a release (-mp)
653  -U      update proto area in the parent
654  -V VERS set the build version string to VERS
655  -f      find unreferenced files
656  -i      do an incremental build (no "make clobber")
657  -l      do "make lint" in $LINTDIRS (default: $SRC y)
658  -m      send mail to $MAILTO at end of build
659  -n      do not do a bringover
660  -p      create packages
661  -r      check ELF runtime attributes in the proto area
662  -t      build and use the tools in $SRC/tools (default setting)
663  +t      Use the build tools in $ONBLD_TOOLS/bin
664  -u      update proto_list_$MACH and friends in the parent workspace;
665  when used with -f, also build an unrefmaster.out in the parent
666  -w      report on differences between previous and current proto areas
667 '
668 #
669 #   A log file will be generated under the name $LOGFILE
670 #   for partially completed build and log.`date +%F`
671 #   in the same directory for fully completed builds.
672 #

674 # default values for low-level FLAGS; G I R are group FLAGS
675 A_FLAG=n
676 C_FLAG=n
677 D_FLAG=n
678 F_FLAG=n
```

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

2

```
679 f_FLAG=n
680 i_FLAG=n; i_CMD_LINE_FLAG=n
681 l_FLAG=n
682 M_FLAG=n
683 m_FLAG=n
684 N_FLAG=n
685 n_FLAG=n
686 p_FLAG=n
687 r_FLAG=n
688 t_FLAG=y
689 U_FLAG=n
690 u_FLAG=n
691 V_FLAG=n
692 w_FLAG=n
693 W_FLAG=n
694 #
695 build_ok=y
696 build_extras_ok=y

698 #
699 # examine arguments
700 #

702 OPTIND=1
703 while getopts +intV:W FLAG
704 do
705     case $FLAG in
706         i ) i_FLAG=y; i_CMD_LINE_FLAG=y
707             ;;
708         n ) n_FLAG=y
709             ;;
710         +t ) t_FLAG=n
711             ;;
712         V ) V_FLAG=y
713             V_ARG="$OPTARG"
714             ;;
715         W ) W_FLAG=y
716             ;;
717         \? ) echo "$USAGE"
718             exit 1
719             ;;
720         esac
721 done

723 # correct argument count after options
724 shift `expr $OPTIND - 1`

726 # test that the path to the environment-setting file was given
727 if [ $# -ne 1 ]; then
728     echo "$USAGE"
729     exit 1
730 fi

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

738 #
739 # force locale to C
740 LANG=C;         export LANG
741 LC_ALL=C;       export LC_ALL
742 #endif /* ! codereview */
743 LC_COLLATE=C;   export LC_COLLATE
744 LC_CTYPE=C;     export LC_CTYPE
```

```

745 LC_MESSAGES=C; export LC_MESSAGES
746 LC_MONETARY=C; export LC_MONETARY
747 LC_NUMERIC=C; export LC_NUMERIC
748 LC_TIME=C; export LC_TIME

750 # clear environment variables we know to be bad for the build
751 unset LD_OPTIONS
752 unset LD_AUDIT LD_AUDIT_32 LD_AUDIT_64
753 unset LD_BIND_NOW LD_BIND_NOW_32 LD_BIND_NOW_64
754 unset LD_BREADTH LD_BREADTH_32 LD_BREADTH_64
755 unset LD_CONFIG LD_CONFIG_32 LD_CONFIG_64
756 unset LD_DEBUG LD_DEBUG_32 LD_DEBUG_64
757 unset LD_DEMANGLE LD_DEMANGLE_32 LD_DEMANGLE_64
758 unset LD_FLAGS LD_FLAGS_32 LD_FLAGS_64
759 unset LD_LIBRARY_PATH LD_LIBRARY_PATH_32 LD_LIBRARY_PATH_64
760 unset LD_LOADFLTR LD_LOADFLTR_32 LD_LOADFLTR_64
761 unset LD_NOAUDIT LD_NOAUDIT_32 LD_NOAUDIT_64
762 unset LD_NOAUXFLTR LD_NOAUXFLTR_32 LD_NOAUXFLTR_64
763 unset LD_NOCONFIG LD_NOCONFIG_32 LD_NOCONFIG_64
764 unset LD_NODIRCONFIG LD_NODIRCONFIG_32 LD_NODIRCONFIG_64
765 unset LD_NODIRECT LD_NODIRECT_32 LD_NODIRECT_64
766 unset LD_NOLAZYLOAD LD_NOLAZYLOAD_32 LD_NOLAZYLOAD_64
767 unset LD_NOOBJALTER LD_NOOBJALTER_32 LD_NOOBJALTER_64
768 unset LD_NOVERSION LD_NOVERSION_32 LD_NOVERSION_64
769 unset LD_ORIGIN LD_ORIGIN_32 LD_ORIGIN_64
770 unset LD_PRELOAD LD_PRELOAD_32 LD_PRELOAD_64
771 unset LD_PROFILE LD_PROFILE_32 LD_PROFILE_64

773 unset CONFIG
774 unset GROUP
775 unset OWNER
776 unset REMOTE
777 unset ENV
778 unset ARCH
779 unset CLASSPATH
780 unset NAME

782 #
783 # To get ONBLD_TOOLS from the environment, it must come from the env file.
784 # If it comes interactively, it is generally TOOLS_PROTO, which will be
785 # clobbered before the compiler version checks, which will therefore fail.
786 #
787 unset ONBLD_TOOLS

789 #
790 # Setup environmental variables
791 #
792 if [ -f /etc/nightly.conf ]; then
793 . /etc/nightly.conf
794 fi

796 if [ -f $1 ]; then
797 if [[ $1 = /* ]]; then
798 . $1
799 else
800 . ./$1
801 fi
802 else
803 if [ -f $OPTHOME/onbld/env/$1 ]; then
804 . $OPTHOME/onbld/env/$1
805 else
806 echo "Cannot find env file as either $1 or $OPTHOME/onbld/env/$1
807 exit 1
808 fi
809 fi

```

```

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

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

825 #
826 # place ourselves in a new task, respecting BUILD_PROJECT if set.
827 #
828 if [ -z "$BUILD_PROJECT" ]; then
829 /usr/bin/newtask -c $$
830 else
831 /usr/bin/newtask -c $$ -p $BUILD_PROJECT
832 fi

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

837 #
838 # See if NIGHTLY_OPTIONS is set
839 #
840 if [ "$NIGHTLY_OPTIONS" = "" ]; then
841 NIGHTLY_OPTIONS="-aBm"
842 fi

844 #
845 # If BRINGOVER_WS was not specified, let it default to CLONE_WS
846 #
847 if [ "$BRINGOVER_WS" = "" ]; then
848 BRINGOVER_WS=$CLONE_WS
849 fi

851 #
852 # If BRINGOVER_FILES was not specified, default to usr
853 #
854 if [ "$BRINGOVER_FILES" = "" ]; then
855 BRINGOVER_FILES="usr"
856 fi

858 check_closed_bins

860 #
861 # Note: changes to the option letters here should also be applied to the
862 # bldenv script. 'd' is listed for backward compatibility.
863 #
864 NIGHTLY_OPTIONS=-${NIGHTLY_OPTIONS#-}
865 OPTIND=1
866 while getopts +ABCDdFfGIlMmNnpRrtUuwW FLAG $NIGHTLY_OPTIONS
867 do
868 case $FLAG in
869 A ) A_FLAG=y
870 ;;
871 B ) D_FLAG=y
872 ;; # old version of D
873 C ) C_FLAG=y
874 ;;
875 D ) D_FLAG=y
876 ;;

```

```

877 F ) F_FLAG=y
878 ;;
879 f ) f_FLAG=y
880 ;;
881 G ) u_FLAG=y
882 ;;
883 I ) m_FLAG=y
884 p_FLAG=y
885 u_FLAG=y
886 ;;
887 i ) i_FLAG=y
888 ;;
889 l ) l_FLAG=y
890 ;;
891 M ) M_FLAG=y
892 ;;
893 m ) m_FLAG=y
894 ;;
895 N ) N_FLAG=y
896 ;;
897 n ) n_FLAG=y
898 ;;
899 p ) p_FLAG=y
900 ;;
901 R ) m_FLAG=y
902 p_FLAG=y
903 ;;
904 r ) r_FLAG=y
905 ;;
906 +t ) t_FLAG=n
907 ;;
908 U ) if [ -z "${PARENT_ROOT}" ]; then
909     echo "PARENT_ROOT must be set if the U flag is" \
910         "present in NIGHTLY_OPTIONS."
911     exit 1
912 fi
913 NIGHTLY_PARENT_ROOT=$PARENT_ROOT
914 if [ -n "${PARENT_TOOLS_ROOT}" ]; then
915     NIGHTLY_PARENT_TOOLS_ROOT=$PARENT_TOOLS_ROOT
916 fi
917 U_FLAG=y
918 ;;
919 u ) u_FLAG=y
920 ;;
921 w ) w_FLAG=y
922 ;;
923 W ) W_FLAG=y
924 ;;
925 \ ? ) echo "$USAGE"
926     exit 1
927 ;;
928 esac
929 done

931 if [ $ISUSER -ne 0 ]; then
932     # Set default value for STAFFER, if needed.
933     if [ -z "$STAFFER" -o "$STAFFER" = "nobody" ]; then
934         STAFFER="/usr/xpg4/bin/id -un"
935         export STAFFER
936     fi
937 fi

939 if [ -z "$MAILTO" -o "$MAILTO" = "nobody" ]; then
940     MAILTO=$STAFFER
941     export MAILTO
942 fi

```

```

944 PATH="$OPTHOME/onbld/bin:$OPTHOME/onbld/bin/${MACH}:/usr/ccs/bin"
945 PATH="$PATH:$OPTHOME/SUNWspro/bin:/usr/bin:/usr/sbin:/usr/uch"
946 PATH="$PATH:/usr/openwin/bin:/usr/sfw/bin:/opt/sfw/bin:."
947 export PATH

949 # roots of source trees, both relative to $SRC and absolute.
950 relsrkdirs="."
951 absrkdirs="$SRC"

953 PROTOCMPTERSE="protocmp.terse -gu"
954 POUND_SIGN="#"
955 # have we set RELEASE_DATE in our env file?
956 if [ -z "$RELEASE_DATE" ]; then
957     RELEASE_DATE=$(LC_ALL=C date +%B %Y)
958 fi
959 BUILD_DATE=$(LC_ALL=C date +%Y-%b-%d)
960 BASEWSDIR=$(basename $CODEMGR_WS)
961 DEV_CM="\@(#)SunOS Internal Development: $LOGNAME $BUILD_DATE [$BASEWSDIR]\\""

963 # we export POUND_SIGN, RELEASE_DATE and DEV_CM to speed up the build process
964 # by avoiding repeated shell invocations to evaluate Makefile.master
965 # definitions.
966 export POUND_SIGN RELEASE_DATE DEV_CM

968 maketype="distributed"
969 if [[ -z "$MAKE" ]]; then
970     MAKE=gmake
971 elif [[ ! -x "$MAKE" ]]; then
972     echo "\$MAKE is set to garbage in the environment"
973     exit 1
974 fi
975 # get the dmake version string alone
976 DMAKE_VERSION=$( $MAKE -v )
977 DMAKE_VERSION=${DMAKE_VERSION#*: }
978 # focus in on just the dotted version number alone
979 DMAKE_MAJOR=$( echo $DMAKE_VERSION | \
980     sed -e 's/.*\([^\.]*\.[^\ ]*\).*$/\1/' )
981 # extract the second (or final) integer
982 DMAKE_MINOR=${DMAKE_MAJOR#*.}
983 DMAKE_MINOR=${DMAKE_MINOR%.*}
984 # extract the first integer
985 DMAKE_MAJOR=${DMAKE_MAJOR%.*}
986 CHECK_DMAKE=${CHECK_DMAKE:-y}
987 # x86 was built on the 12th, sparc on the 13th.
988 if [ "$CHECK_DMAKE" = "y" -a \
989     "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/12" -a \
990     "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/13" -a \
991     "$DMAKE_MAJOR" -lt 7 -o \
992     "$DMAKE_MAJOR" -eq 7 -a "$DMAKE_MINOR" -lt 4 \ ] ]; then
993     if [ -z "$DMAKE_VERSION" ]; then
994         echo "$MAKE is missing."
995         exit 1
996     fi
997     echo `whence $MAKE` version is:
998     echo " $${DMAKE_VERSION}"
999     cat <<EOF

1001 This version may not be safe for use, if you really want to use this version
1002 anyway add the following to your environment to disable this check:

1004 CHECK_DMAKE=n
1005 EOF
1006     exit 1
1007 fi
1008 export PATH

```

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

7

```
1009 export MAKE

1011 if [ "${SUNWSPRO}" != "" ]; then
1012   PATH="${SUNWSPRO}/bin:$PATH"
1013   export PATH
1014 fi

1016 hostname=$(uname -n)
1017 if [[ $DMAKE_MAX_JOBS != +([0-9]) || $DMAKE_MAX_JOBS -eq 0 ]]
1018 then
1019   maxjobs=
1020   if [[ -f $HOME/.make.machines ]]
1021   then
1022     # Note: there is a hard tab and space character in the []s
1023     # below.
1024     egrep -i "^[ \t]*$hostname[ \t].*" \
1025       $HOME/.make.machines | read host jobs
1026     maxjobs=${jobs##*=}
1027   fi

1029   if [[ $maxjobs != +([0-9]) || $maxjobs -eq 0 ]]
1030   then
1031     # default
1032     maxjobs=4
1033   fi

1035   export DMAKE_MAX_JOBS=$maxjobs
1036 fi

1038 DMAKE_MODE=parallel;
1039 export DMAKE_MODE

1041 if [ -z "${ROOT}" ]; then
1042   echo "ROOT must be set."
1043   exit 1
1044 fi

1046 #
1047 # if -V flag was given, reset VERSION to V_ARG
1048 #
1049 if [ "$V_FLAG" = "y" ]; then
1050   VERSION=$V_ARG
1051 fi

1053 TMPDIR="/tmp/nightly.tmpdir.$$"
1054 export TMPDIR
1055 rm -rf ${TMPDIR}
1056 mkdir -p $TMPDIR || exit 1
1057 chmod 777 $TMPDIR

1059 #
1060 # Keep elfsign's use of pkcs11_softtoken from looking in the user home
1061 # directory, which doesn't always work. Needed until all build machines
1062 # have the fix for 6271754
1063 #
1064 SOFTOKEN_DIR=$TMPDIR
1065 export SOFTOKEN_DIR

1067 #
1068 # Tools should only be built non-DEBUG. Keep track of the tools proto
1069 # area path relative to $TOOLS, because the latter changes in an
1070 # export build.
1071 #
1072 # TOOLS_PROTO is included below for builds other than usr/src/tools
1073 # that look for this location. For usr/src/tools, this will be
1074 # overridden on the $MAKE command line in build_tools().
```

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

8

```
1075 #
1076 TOOLS=${SRC}/tools
1077 TOOLS_PROTO_REL=proto/root_${MACH}-nd
1078 TOOLS_PROTO=${TOOLS}/${TOOLS_PROTO_REL}; export TOOLS_PROTO

1080 unset CFLAGS LD_LIBRARY_PATH LDFLAGS

1082 # create directories that are automatically removed if the nightly script
1083 # fails to start correctly
1084 function newdir {
1085   dir=$1
1086   toadd=
1087   while [ ! -d $dir ]; do
1088     toadd="$dir $toadd"
1089     dir='dirname $dir'
1090   done
1091   torm=
1092   newlist=
1093   for dir in $toadd; do
1094     if staffer mkdir $dir; then
1095       newlist="$ISUSER $dir $newlist"
1096       torm="$dir $torm"
1097     else
1098       [ -z "$torm" ] || staffer rmdir $torm
1099       return 1
1100     fi
1101   done
1102   newdirlist="$newlist $newdirlist"
1103   return 0
1104 }
1105 newdirlist=

1107 [ -d $CODEMGR_WS ] || newdir $CODEMGR_WS || exit 1

1109 # since this script assumes the build is from full source, it nullifies
1110 # variables likely to have been set by a "ws" script; nullification
1111 # confines the search space for headers and libraries to the proto area
1112 # built from this immediate source.
1113 ENVLDLIBS1=
1114 ENVLDLIBS2=
1115 ENVLDLIBS3=
1116 ENVCPPFLAGS1=
1117 ENVCPPFLAGS2=
1118 ENVCPPFLAGS3=
1119 ENVCPPFLAGS4=
1120 PARENT_ROOT=

1122 export ENVLDLIBS3 ENVCPPFLAGS1 ENVCPPFLAGS2 ENVCPPFLAGS3 ENVCPPFLAGS4 \
1123   ENVLDLIBS1 ENVLDLIBS2 PARENT_ROOT

1125 PKGARCHIVE_ORIG=$PKGARCHIVE

1127 #
1128 # Juggle the logs and optionally send mail on completion.
1129 #

1131 function logshuffle {
1132   LLOG="$ATLOG/log.`date +%F.%H:%M`"
1133   if [ -f $LLOG -o -d $LLOG ]; then
1134     LLOG=$LLOG.$$
1135   fi
1136   mkdir $LLOG
1137   export LLOG

1139   if [ "$build_ok" = "y" ]; then
1140     mv $ATLOG/proto_list_${MACH} $LLOG
```

```

1142         if [ -f $ATLOG/proto_list_tools_${MACH} ]; then
1143             mv $ATLOG/proto_list_tools_${MACH} $LLOG
1144         fi

1146         if [ -f $TMPDIR/wsdiff.results ]; then
1147             mv $TMPDIR/wsdiff.results $LLOG
1148         fi

1150         if [ -f $TMPDIR/wsdiff-nd.results ]; then
1151             mv $TMPDIR/wsdiff-nd.results $LLOG
1152         fi
1153     fi

1155     #
1156     # Now that we're about to send mail, it's time to check the noise
1157     # file. In the event that an error occurs beyond this point, it will
1158     # be recorded in the nightly.log file, but nowhere else. This would
1159     # include only errors that cause the copying of the noise log to fail
1160     # or the mail itself not to be sent.
1161     #

1163     exec >>$LOGFILE 2>&1
1164     if [ -s $build_noise_file ]; then
1165         echo "\n=== Nightly build noise ===\n" |
1166             tee -a $LOGFILE >>$mail_msg_file
1167         cat $build_noise_file >>$LOGFILE
1168         cat $build_noise_file >>$mail_msg_file
1169         echo | tee -a $LOGFILE >>$mail_msg_file
1170     fi
1171     rm -f $build_noise_file

1173     case "$build_ok" in
1174         y)
1175             state=Completed
1176             ;;
1177         i)
1178             state=Interrupted
1179             ;;
1180         *)
1181             state=Failed
1182             ;;
1183     esac

1185     if [[ $state != "Interrupted" && $build_extras_ok != "y" ]]; then
1186         state=Failed
1187     fi

1189     NIGHTLY_STATUS=$state
1190     export NIGHTLY_STATUS

1192     run_hook POST_NIGHTLY $state
1193     run_hook SYS_POST_NIGHTLY $state

1195     #
1196     # mailx(1) sets From: based on the -r flag
1197     # if it is given.
1198     #
1199     mailx_r=
1200     if [[ -n "${MAILFROM}" ]]; then
1201         mailx_r="-r ${MAILFROM}"
1202     fi

1204     cat $build_time_file $build_environ_file $mail_msg_file \
1205         > ${LLOG}/mail_msg
1206     if [ "$m_FLAG" = "y" ]; then

```

```

1207         cat ${LLOG}/mail_msg | /usr/bin/mailx ${mailx_r} -s \
1208             "Nightly ${MACH} Build of 'basename ${CODEMGR_WS}' ${state}." \
1209             ${MAILTO}
1210     fi

1212     if [ "$u_FLAG" = "y" -a "$build_ok" = "y" ]; then
1213         staffer cp ${LLOG}/mail_msg $PARENT_WS/usr/src/mail_msg-${MACH}
1214         staffer cp $LOGFILE $PARENT_WS/usr/src/nightly-${MACH}.log
1215     fi

1217     mv $LOGFILE $LLOG
1218 }

1220 #
1221 # Remove the locks and temporary files on any exit
1222 #
1223 function cleanup {
1224     logshuffler

1226     [ -z "$lockfile" ] || staffer rm -f $lockfile
1227     [ -z "$atloglockfile" ] || rm -f $atloglockfile
1228     [ -z "$ulockfile" ] || staffer rm -f $ulockfile
1229     [ -z "$Ulockfile" ] || rm -f $Ulockfile

1231     set -- $newdirlist
1232     while [ $# -gt 0 ]; do
1233         ISUSER=$1 staffer rmdir $2
1234         shift; shift
1235     done
1236     rm -rf $TMPDIR
1237 }

1239 function cleanup_signal {
1240     build_ok=i
1241     # this will trigger cleanup(), above.
1242     exit 1
1243 }

1245 trap cleanup 0
1246 trap cleanup_signal 1 2 3 15

1248 #
1249 # Generic lock file processing -- make sure that the lock file doesn't
1250 # exist. If it does, it should name the build host and PID. If it
1251 # doesn't, then make sure we can create it. Clean up locks that are
1252 # known to be stale (assumes host name is unique among build systems
1253 # for the workspace).
1254 #
1255 function create_lock {
1256     lockf=$1
1257     lockvar=$2

1259     ldir='dirname $lockf'
1260     [ -d $ldir ] || newdir $ldir || exit 1
1261     eval $lockvar=$lockf

1263     while ! staffer ln -s $hostname.$STAFFER.$$ $lockf 2> /dev/null; do
1264         basews='basename $CODEMGR_WS'
1265         ls -l $lockf | nawk '{print $NF}' | IFS=. read host user pid
1266         if [ "$host" != "$hostname" ]; then
1267             echo "$MACH build of $basews apparently" \
1268                 "already started by $user on $host as $pid."
1269             exit 1
1270         elif kill -s 0 $pid 2>/dev/null; then
1271             echo "$MACH build of $basews already started" \
1272                 "by $user as $pid."

```



```

1273         exit 1
1274     else
1275         # stale lock; clear it out and try again
1276         rm -f $lockf
1277     fi
1278 done
1279 }

1281 #
1282 # Return the list of interesting proto areas, depending on the current
1283 # options.
1284 #
1285 function allprotos {
1286     typeset roots="$ROOT"

1288     if [[ "$F_FLAG" = n && "$MULTI_PROTO" = yes ]]; then
1289         roots="$roots $ROOT-nd"
1290     fi

1292     echo $roots
1293 }

1295 # Ensure no other instance of this script is running on this host.
1296 # LOCKNAME can be set in <env_file>, and is by default, but is not
1297 # required due to the use of $ATLOG below.
1298 if [ -n "$LOCKNAME" ]; then
1299     create_lock /tmp/$LOCKNAME "lockfile"
1300 fi
1301 #
1302 # Create from one, two, or three other locks:
1303 # $ATLOG/nightly.lock
1304 # - protects against multiple builds in same workspace
1305 # $PARENT_WS/usr/src/nightly.$MACH.lock
1306 # - protects against multiple 'u' copy-backs
1307 # $NIGHTLY_PARENT_ROOT/nightly.lock
1308 # - protects against multiple 'U' copy-backs
1309 #
1310 # Overriding ISUSER to 1 causes the lock to be created as root if the
1311 # script is run as root. The default is to create it as $STAFFER.
1312 ISUSER=1 create_lock $ATLOG/nightly.lock "atloglockfile"
1313 if [ "$u_FLAG" = "y" ]; then
1314     create_lock $PARENT_WS/usr/src/nightly.$MACH.lock "unlockfile"
1315 fi
1316 if [ "$U_FLAG" = "y" ]; then
1317     # NIGHTLY_PARENT_ROOT is written as root if script invoked as root.
1318     ISUSER=1 create_lock $NIGHTLY_PARENT_ROOT/nightly.lock "Ulockfile"
1319 fi

1321 # Locks have been taken, so we're doing a build and we're committed to
1322 # the directories we may have created so far.
1323 newdirlist=

1325 #
1326 # Create mail_msg_file
1327 #
1328 mail_msg_file="${TMPDIR}/mail_msg"
1329 touch $mail_msg_file
1330 build_time_file="${TMPDIR}/build_time"
1331 build_envIRON_file="${TMPDIR}/build_envIRON"
1332 touch $build_envIRON_file
1333 #
1334 # Move old LOGFILE aside
1335 # ATLOG directory already made by 'create_lock' above
1336 #
1337 if [ -f $LOGFILE ]; then
1338     mv -f $LOGFILE ${LOGFILE}-

```

```

1339 fi
1340 #
1341 # Build OsNet source
1342 #
1343 START_DATE=`date`
1344 SECONDS=0
1345 echo "\n==== Nightly $maketype build started:  $START_DATE ==== \
1346 | tee -a $LOGFILE > $build_time_file

1348 echo "\nBuild project: $build_project\nBuild taskid:  $build_taskid" | \
1349 tee -a $mail_msg_file >> $LOGFILE

1351 # make sure we log only to the nightly build file
1352 build_noise_file="${TMPDIR}/build_noise"
1353 exec </dev/null >$build_noise_file 2>&1

1355 run_hook SYS_PRE_NIGHTLY
1356 run_hook PRE_NIGHTLY

1358 echo "\n==== list of environment variables ==== \n" >> $LOGFILE
1359 env >> $LOGFILE

1361 echo "\n==== Nightly argument issues ==== \n" | tee -a $mail_msg_file >> $LOGFILE

1363 if [ "$N_FLAG" = "y" ]; then
1364     if [ "$p_FLAG" = "y" ]; then
1365         cat <<EOF | tee -a $mail_msg_file >> $LOGFILE
1366 WARNING: the p option (create packages) is set, but so is the N option (do
1367 not run protocmp); this is dangerous; you should unset the N option
1368 EOF
1369     else
1370         cat <<EOF | tee -a $mail_msg_file >> $LOGFILE
1371 Warning: the N option (do not run protocmp) is set; it probably shouldn't be
1372 EOF
1373     fi
1374     echo "" | tee -a $mail_msg_file >> $LOGFILE
1375 fi

1377 if [ "$D_FLAG" = "n" -a "$l_FLAG" = "y" ]; then
1378     #
1379     # In the past we just complained but went ahead with the lint
1380     # pass, even though the proto area was built non-DEBUG. It's
1381     # unlikely that non-DEBUG headers will make a difference, but
1382     # rather than assuming it's a safe combination, force the user
1383     # to specify a DEBUG build.
1384     #
1385     echo "WARNING: DEBUG build not requested; disabling lint.\n" \
1386         | tee -a $mail_msg_file >> $LOGFILE
1387     l_FLAG=n
1388 fi

1390 if [ "$f_FLAG" = "y" ]; then
1391     if [ "$i_FLAG" = "y" ]; then
1392         echo "WARNING: the -f flag cannot be used during incremental" \
1393             "builds; ignoring -f\n" | tee -a $mail_msg_file >> $LOGFILE
1394         f_FLAG=n
1395     fi
1396     if [ "${l_FLAG}${p_FLAG}" != "yy" ]; then
1397         echo "WARNING: the -f flag requires -l, and -p;" \
1398             "ignoring -f\n" | tee -a $mail_msg_file >> $LOGFILE
1399         f_FLAG=n
1400     fi
1401 fi

1403 if [ "$w_FLAG" = "y" -a ! -d $ROOT ]; then
1404     echo "WARNING: -w specified, but $ROOT does not exist;" \

```

```

1405     "ignoring -w\n" | tee -a $mail_msg_file >> $LOGFILE
1406     w_FLAG=n
1407 fi

1409 if [ "$t_FLAG" = "n" ]; then
1410     #
1411     # We're not doing a tools build, so make sure elfsign(1) is
1412     # new enough to safely sign non-crypto binaries. We test
1413     # debugging output from elfsign to detect the old version.
1414     #
1415     newelfsigntest=\SUNW_CRYPT0_DEBUG=stderr /usr/bin/elfsign verify \
1416     -e /usr/lib/security/pkcs11_softtoken.so.1 2>&1 \
1417     | egrep algorithmOID`
1418     if [ -z "$newelfsigntest" ]; then
1419         echo "WARNING: /usr/bin/elfsign out of date;" \
1420             "will only sign crypto modules\n" | \
1421             tee -a $mail_msg_file >> $LOGFILE
1422         export ELFSIGN_OBJECT=true
1423     elif [ "$VERIFY_ELFSIGN" = "y" ]; then
1424         echo "WARNING: VERIFY_ELFSIGN=y requires" \
1425             "the -t flag; ignoring VERIFY_ELFSIGN\n" | \
1426             tee -a $mail_msg_file >> $LOGFILE
1427     fi
1428 fi

1430 case $MULTI_PROTO in
1431 yes|no) ;;
1432 *)
1433     echo "WARNING: MULTI_PROTO is \"$MULTI_PROTO\"; " \
1434         "should be \"yes\" or \"no\"." | tee -a $mail_msg_file >> $LOGFILE
1435     echo "Setting MULTI_PROTO to \"no\".\n" | \
1436         tee -a $mail_msg_file >> $LOGFILE
1437     export MULTI_PROTO=no
1438     ;;
1439 esac

1441 echo "\n=== Build version ===\n" | tee -a $mail_msg_file >> $LOGFILE
1442 echo $VERSION | tee -a $mail_msg_file >> $LOGFILE

1444 # Save the current proto area if we're comparing against the last build
1445 if [ "$w_FLAG" = "y" -a -d "$ROOT" ]; then
1446     if [ -d "$ROOT.prev" ]; then
1447         rm -rf $ROOT.prev
1448     fi
1449     mv $ROOT $ROOT.prev
1450 fi

1452 # Same for non-DEBUG proto area
1453 if [ "$w_FLAG" = "y" -a "$MULTI_PROTO" = yes -a -d "$ROOT-nd" ]; then
1454     if [ -d "$ROOT-nd.prev" ]; then
1455         rm -rf $ROOT-nd.prev
1456     fi
1457     mv $ROOT-nd $ROOT-nd.prev
1458 fi

1460 #
1461 # Echo the SCM type of the parent workspace, this can't just be which_scm
1462 # as that does not know how to identify various network repositories.
1463 #
1464 function parent_wstype {
1465     typeset scm_type junk

1467     CODEMGR_WS="$BRINGOVER_WS" "$WHICH_SCM" 2>/dev/null \
1468     | read scm_type junk
1469     if [[ -z "$scm_type" || "$scm_type" == unknown ]]; then
1470         # Probe BRINGOVER_WS to determine its type

```

```

1471         if [[ $BRINGOVER_WS == ssh://* ]]; then
1472             scm_type="mercurial"
1473         elif [[ $BRINGOVER_WS == http://* ]] && \
1474             wget -q -O- --save-headers "$BRINGOVER_WS/?cmd=heads" | \
1475             egrep -s "application/mercurial" 2> /dev/null; then
1476             scm_type="mercurial"
1477         else
1478             scm_type="none"
1479         fi
1480     fi

1482     # fold both unsupported and unrecognized results into "none"
1483     case "$scm_type" in
1484     mercurial)
1485         ;;
1486     *)
1487         scm_type=none
1488     esac
1489     ;;
1490     echo $scm_type
1491 }

1493 # Echo the SCM types of $CODEMGR_WS and $BRINGOVER_WS
1494 function child_wstype {
1495     typeset scm_type junk

1497     # Probe CODEMGR_WS to determine its type
1498     if [[ -d $CODEMGR_WS ]]; then
1499         $WHICH_SCM | read scm_type junk || exit 1
1500     fi

1502     case "$scm_type" in
1503     none|git|mercurial)
1504         ;;
1505     *)
1506         scm_type=none
1507     esac
1508     ;;
1509     echo $scm_type
1510 }

1512 SCM_TYPE=$(child_wstype)

1514 #
1515 #     Decide whether to clobber
1516 #
1517 if [ "$i_FLAG" = "n" -a -d "$SRC" ]; then
1518     echo "\n=== Make clobber at 'date' ===\n" >> $LOGFILE

1520     cd $SRC
1521     # remove old clobber file
1522     rm -f $SRC/clobber.out
1523     rm -f $SRC/clobber-${MACH}.out

1525     # Remove all .make.state* files, just in case we are restarting
1526     # the build after having interrupted a previous 'make clobber'.
1527     find . \( -name SCCS -o -name .hg -o -name .svn -o -name .git \
1528         -o -name 'interfaces.*' \) -prune \
1529         -o -name '*.make.*' -print | xargs rm -f

1531     $MAKE -ek clobber 2>&1 | tee -a $SRC/clobber-${MACH}.out >> $LOGFILE
1532     echo "\n=== Make clobber ERRORS ===\n" >> $mail_msg_file
1533     grep "$MAKE:" $SRC/clobber-${MACH}.out |
1534         egrep -v "Ignoring unknown host" | \
1535         tee $TMPDIR/clobber_errs >> $mail_msg_file

```

```

1537     if [[ -s $TMPDIR/clobber_errs ]]; then
1538         build_extras_ok=n
1539     fi

1541     if [[ "$t_FLAG" = "y" ]]; then
1542         echo "\n=== Make tools clobber at `date` ===\n" >> $LOGFILE
1543         cd ${TOOLS}
1544         rm -f ${TOOLS}/clobber-${MACH}.out
1545         $MAKE TOOLS_PROTO=${TOOLS_PROTO} -ek clobber 2>&1 | \
1546             tee -a ${TOOLS}/clobber-${MACH}.out >> $LOGFILE
1547         echo "\n=== Make tools clobber ERRORS ===\n" \
1548             >> $mail_msg_file
1549         grep "MAKE:" ${TOOLS}/clobber-${MACH}.out \
1550             >> $mail_msg_file
1551         if (( $? == 0 )); then
1552             build_extras_ok=n
1553         fi
1554         rm -rf ${TOOLS_PROTO}
1555         mkdir -p ${TOOLS_PROTO}
1556     fi

1558     typeset roots=$(allprotos)
1559     echo "\n\nClearing $roots" >> "$LOGFILE"
1560     rm -rf $roots

1562     # Get back to a clean workspace as much as possible to catch
1563     # problems that only occur on fresh workspaces.
1564     # Remove all .make.state* files, libraries, and .o's that may
1565     # have been omitted from clobber. A couple of libraries are
1566     # under source code control, so leave them alone.
1567     # We should probably blow away temporary directories too.
1568     cd $SRC
1569     find $srcrcdirs \( -name SCCS -o -name .hg -o -name .svn \
1570         -o -name .git -o -name 'interfaces.*' \) -prune -o \
1571         \( -name '.make.*' -o -name 'lib*.a' -o -name 'lib*.so*' -o \
1572         -name '*.o' \) -print | \
1573         grep -v 'tools/ctf/dwarf/.*/libdwarf' | xargs rm -f
1574 else
1575     echo "\n=== No clobber at `date` ===\n" >> $LOGFILE
1576 fi

1578 type bringover_mercurial > /dev/null 2>&1 || function bringover_mercurial {
1579     typeset -x PATH=$PATH

1581     # If the repository doesn't exist yet, then we want to populate it.
1582     if [[ ! -d $CODEMGR_WS/.hg ]]; then
1583         staffer hg init $CODEMGR_WS
1584         staffer echo "[paths]" > $CODEMGR_WS/.hg/hgrc
1585         staffer echo "default=$BRINGOVER_WS" >> $CODEMGR_WS/.hg/hgrc
1586         touch $TMPDIR/new_repository
1587     fi

1589     typeset -x HGMERGE="/bin/false"

1591     #
1592     # If the user has changes, regardless of whether those changes are
1593     # committed, and regardless of whether those changes conflict, then
1594     # we'll attempt to merge them either implicitly (uncommitted) or
1595     # explicitly (committed).
1596     #
1597     # These are the messages we'll use to help clarify mercurial output
1598     # in those cases.
1599     #
1600     typeset mergefailmsg="\
1601 ***\n\
1602 *** nightly was unable to automatically merge your changes. You should\n\

```

```

1603 *** redo the full merge manually, following the steps outlined by mercurial\n\
1604 *** above, then restart nightly.\n\
1605 ***\n"
1606         typeset mergepassmsg="\
1607 ***\n\
1608 *** nightly successfully merged your changes. This means that your working\n\
1609 *** directory has been updated, but those changes are not yet committed.\n\
1610 *** After nightly completes, you should validate the results of the merge,\n\
1611 *** then use hg commit manually.\n\
1612 ***\n"

1614     #
1615     # For each repository in turn:
1616     #
1617     # 1. Do the pull. If this fails, dump the output and bail out.
1618     #
1619     # 2. If the pull resulted in an extra head, do an explicit merge.
1620     #    If this fails, dump the output and bail out.
1621     #
1622     # Because we can't rely on Mercurial to exit with a failure code
1623     # when a merge fails (Mercurial issue #186), we must grep the
1624     # output of pull/merge to check for attempted and/or failed merges.
1625     #
1626     # 3. If a merge failed, set the message and fail the bringover.
1627     #
1628     # 4. Otherwise, if a merge succeeded, set the message
1629     #
1630     # 5. Dump the output, and any message from step 3 or 4.
1631     #

1633     typeset HG_SOURCE=$BRINGOVER_WS
1634     if [ ! -f $TMPDIR/new_repository ]; then
1635         HG_SOURCE=$TMPDIR/open_bundle.hg
1636         staffer hg --cwd $CODEMGR_WS incoming --bundle $HG_SOURCE \
1637             -v $BRINGOVER_WS > $TMPDIR/incoming_open.out

1639     #
1640     # If there are no incoming changesets, then incoming will
1641     # fail, and there will be no bundle file. Reset the source,
1642     # to allow the remaining logic to complete with no false
1643     # negatives. (Unlike incoming, pull will return success
1644     # for the no-change case.)
1645     #
1646     if (( $? != 0 )); then
1647         HG_SOURCE=$BRINGOVER_WS
1648     fi
1649 fi

1651     staffer hg --cwd $CODEMGR_WS pull -u $HG_SOURCE \
1652         > $TMPDIR/pull_open.out 2>&1
1653     if (( $? != 0 )); then
1654         printf "%s: pull failed as follows:\n\n" "$CODEMGR_WS"
1655         cat $TMPDIR/pull_open.out
1656         if grep "^merging.*failed" $TMPDIR/pull_open.out > /dev/null 2>&
1657             printf "$mergefailmsg"
1658         fi
1659         touch $TMPDIR/bringover_failed
1660         return
1661     fi

1663     if grep "not updating" $TMPDIR/pull_open.out > /dev/null 2>&1; then
1664         staffer hg --cwd $CODEMGR_WS merge \
1665             >> $TMPDIR/pull_open.out 2>&1
1666         if (( $? != 0 )); then
1667             printf "%s: merge failed as follows:\n\n" \
1668                 "$CODEMGR_WS"

```

```

1669         cat $TMPDIR/pull_open.out
1670         if grep "^merging.*failed" $TMPDIR/pull_open.out \
1671             > /dev/null 2>&1; then
1672             printf "$mergefailmsg"
1673         fi
1674         touch $TMPDIR/bringover_failed
1675         return
1676     fi
1677 fi

1679 printf "updated %s with the following results:\n" "$CODEMGR_WS"
1680 cat $TMPDIR/pull_open.out
1681 if grep "^merging" $TMPDIR/pull_open.out >/dev/null 2>&1; then
1682     printf "$mergepassmsg"
1683 fi
1684 printf "\n"

1686 #
1687 # Per-changeset output is neither useful nor manageable for a
1688 # newly-created repository.
1689 #
1690 if [ -f $TMPDIR/new_repository ]; then
1691     return
1692 fi

1694 printf "\nadded the following changesets to open repository:\n"
1695 cat $TMPDIR/incoming_open.out
1696 }

1698 type bringover_none > /dev/null 2>&1 || function bringover_none {
1699     echo "Couldn't figure out what kind of SCM to use for $BRINGOVER_WS."
1700     touch $TMPDIR/bringover_failed
1701 }

1703 #
1704 # Decide whether to bringover to the codemgr workspace
1705 #
1706 if [ "$n_FLAG" = "n" ]; then
1707     PARENT_SCM_TYPE=$(parent_wstype)

1709     if [[ $SCM_TYPE != none && $SCM_TYPE != $PARENT_SCM_TYPE ]]; then
1710         echo "cannot bringover from $PARENT_SCM_TYPE to $SCM_TYPE, " \
1711             "quitting at `date`. " | tee -a $mail_msg_file >> $LOGFILE
1712         exit 1
1713     fi

1715     run_hook PRE_BRINGOVER

1717     echo "\n==== bringover to $CODEMGR_WS at `date` ==== \n" >> $LOGFILE
1718     echo "\n==== BRINGOVER LOG ==== \n" >> $mail_msg_file

1720     eval "bringover ${PARENT_SCM_TYPE}" 2>&1 |
1721         tee -a $mail_msg_file >> $LOGFILE

1723     if [ -f $TMPDIR/bringover_failed ]; then
1724         rm -f $TMPDIR/bringover_failed
1725         build_ok=n
1726         echo "trouble with bringover, quitting at `date`. " |
1727             tee -a $mail_msg_file >> $LOGFILE
1728         exit 1
1729     fi

1731     #
1732     # It's possible that we used the bringover above to create
1733     # $CODEMGR_WS. If so, then SCM_TYPE was previously "none,"
1734     # but should now be the same as $BRINGOVER_WS.

```

```

1735     #
1736     [[ $SCM_TYPE = none ]] && SCM_TYPE=$PARENT_SCM_TYPE

1738     run_hook POST_BRINGOVER

1740     check_closed_bins

1742 else
1743     echo "\n==== No bringover to $CODEMGR_WS ==== \n" >> $LOGFILE
1744 fi

1746 # Safeguards
1747 [[ -v CODEMGR_WS ]] || fatal_error "Error: Variable CODEMGR_WS not set."
1748 [[ -d "${CODEMGR_WS}" ]] || fatal_error "Error: ${CODEMGR_WS} is not a directory"
1749 [[ -f "${CODEMGR_WS}/usr/src/Makefile" ]] || fatal_error "Error: ${CODEMGR_WS}/u

1751 echo "\n==== Build environment ==== \n" | tee -a $build_envIRON_file >> $LOGFILE

1753 # System
1754 whence uname | tee -a $build_envIRON_file >> $LOGFILE
1755 uname -a 2>&1 | tee -a $build_envIRON_file >> $LOGFILE
1756 echo | tee -a $build_envIRON_file >> $LOGFILE

1758 # make
1759 whence $MAKE | tee -a $build_envIRON_file >> $LOGFILE
1760 $MAKE -v | tee -a $build_envIRON_file >> $LOGFILE
1761 echo "number of concurrent jobs = $DMAKE_MAX_JOBS" |
1762     tee -a $build_envIRON_file >> $LOGFILE

1764 #
1765 # Report the compiler versions.
1766 #

1768 if [[ ! -f $SRC/Makefile ]]; then
1769     build_ok=n
1770     echo "\nUnable to find \"Makefile\" in $SRC." | \
1771         tee -a $build_envIRON_file >> $LOGFILE
1772     exit 1
1773 fi

1775 ( cd $SRC
1776     for target in cc-version cc64-version java-version; do
1777         echo
1778         #
1779         # Put statefile somewhere we know we can write to rather than trip
1780         # over a read-only $srcroot.
1781         #
1782         rm -f $TMPDIR/make-state
1783         export SRC
1784         if $MAKE -K $TMPDIR/make-state -e $target 2>/dev/null; then
1785             continue
1786         fi
1787         touch $TMPDIR/nocompiler
1788     done
1789     echo
1790 ) | tee -a $build_envIRON_file >> $LOGFILE

1792 if [ -f $TMPDIR/nocompiler ]; then
1793     rm -f $TMPDIR/nocompiler
1794     build_ok=n
1795     echo "Aborting due to missing compiler." |
1796         tee -a $build_envIRON_file >> $LOGFILE
1797     exit 1
1798 fi

1800 # as

```

```

1801 whence as | tee -a $build_environ_file >> $LOGFILE
1802 as -V 2>&1 | head -1 | tee -a $build_environ_file >> $LOGFILE
1803 echo | tee -a $build_environ_file >> $LOGFILE

1805 # Check that we're running a capable link-editor
1806 whence ld | tee -a $build_environ_file >> $LOGFILE
1807 LDVER=`ld -v 2>&1`
1808 echo $LDVER | tee -a $build_environ_file >> $LOGFILE
1809 LDVER=`echo $LDVER | sed -e "s/.*-1\\.\\([0-9]*\\).*/\\1/"`
1810 if [ `expr $LDVER < 422` -eq 1 ]; then
1811     echo "The link-editor needs to be at version 422 or higher to build" | \
1812     tee -a $build_environ_file >> $LOGFILE
1813     echo "the latest stuff. Hope your build works." | \
1814     tee -a $build_environ_file >> $LOGFILE
1815 fi

1817 #
1818 # Build and use the workspace's tools if requested
1819 #
1820 if [[ "$t_FLAG" = "y" ]]; then
1821     set_non_debug_build_flags

1823     build_tools ${TOOLS_PROTO}
1824     if (( $? != 0 )); then
1825         build_ok=n
1826     else
1827         use_tools $TOOLS_PROTO
1828     fi
1829 fi

1831 # timestamp the start of the normal build; the findunref tool uses it.
1832 touch $SRC/.build.tstamp

1834 normal_build

1836 ORIG_SRC=$SRC
1837 BINARCHIVE=${CODEMGR_WS}/bin-${MACH}.cpio.Z

1840 #
1841 # There are several checks that need to look at the proto area, but
1842 # they only need to look at one, and they don't care whether it's
1843 # DEBUG or non-DEBUG.
1844 #
1845 if [[ "$MULTI_PROTO" = yes && "$D_FLAG" = n ]]; then
1846     checkroot=$ROOT-nd
1847 else
1848     checkroot=$ROOT
1849 fi

1851 if [ "$build_ok" = "y" ]; then
1852     echo "\n==== Creating protolist system file at `date` ==== \
1853     >> $LOGFILE
1854     protolist $checkroot > $ATLOG/proto_list_${MACH}
1855     echo "==== protolist system file created at `date` ==== \
1856     >> $LOGFILE

1858     if [ "$N_FLAG" != "y" ]; then

1860         E1=
1861         f1=
1862         for f in $f1; do
1863             if [ -f "$f" ]; then
1864                 E1="$E1 -e $f"
1865             fi
1866         done

```

```

1868         E2=
1869         f2=
1870         if [ -d "$SRC/pkg" ]; then
1871             f2="$f2 exceptions/packaging"
1872         fi

1874         for f in $f2; do
1875             if [ -f "$f" ]; then
1876                 E2="$E2 -e $f"
1877             fi
1878         done
1879     fi

1881     if [ "$N_FLAG" != "y" -a -d $SRC/pkg ]; then
1882         echo "\n==== Validating manifests against proto area ==== \
1883         >> $mail_msg_file
1884         ( cd $SRC/pkg ; $MAKE -e protocmp ROOT="$checkroot" ) | \
1885         tee $TMPDIR/protocmp_noise >> $mail_msg_file
1886         if [[ -s $TMPDIR/protocmp_noise ]]; then
1887             build_extras_ok=n
1888         fi
1889     fi

1891     if [ "$N_FLAG" != "y" -a -f "$REF_PROTO_LIST" ]; then
1892         echo "\n==== Impact on proto area ==== \
1893         if [ -n "$E2" ]; then
1894             ELIST=$E2
1895         else
1896             ELIST=$E1
1897         fi
1898         $PROTCMPERSE \
1899             "Files in yesterday's proto area, but not today's:" \
1900             "Files in today's proto area, but not yesterday's:" \
1901             "Files that changed between yesterday and today:" \
1902             ${ELIST} \
1903             -d $REF_PROTO_LIST \
1904             $ATLOG/proto_list_${MACH} \
1905             >> $mail_msg_file
1906     fi
1907 fi

1909 if [[ "$u_FLAG" == "y" && "$build_ok" == "y" && \
1910     "$build_extras_ok" == "y" ]]; then
1911     staffer cp $ATLOG/proto_list_${MACH} \
1912     $PARENT_WS/usr/src/proto_list_${MACH}
1913 fi

1915 # Update parent proto area if necessary. This is done now
1916 # so that the proto area has either DEBUG or non-DEBUG kernels.
1917 # Note that this clears out the lock file, so we can dispense with
1918 # the variable now.
1919 if [ "$U_FLAG" = "y" -a "$build_ok" = "y" ]; then
1920     echo "\n==== Copying proto area to $NIGHTLY_PARENT_ROOT ==== \
1921     >> $mail_msg_file
1922     rm -rf $NIGHTLY_PARENT_ROOT/*
1923     unset Ulockfile
1924     mkdir -p $NIGHTLY_PARENT_ROOT
1925     if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
1926         ( cd $ROOT; tar cf - . |
1927         ( cd $NIGHTLY_PARENT_ROOT; umask 0; tar xpf - ) ) 2>&1 |
1928         tee -a $mail_msg_file >> $LOGFILE
1929     fi
1930     if [[ "$MULTI_PROTO" = yes && "$F_FLAG" = n ]]; then
1931         rm -rf $NIGHTLY_PARENT_ROOT-nd/*
1932         mkdir -p $NIGHTLY_PARENT_ROOT-nd

```

```

1933         cd $ROOT-nd
1934         ( tar cf - . |
1935         ( cd $NIGHTLY_PARENT_ROOT-nd; umask 0; tar xpf - ) ) 2>&1 |
1936         tee -a $mail_msg_file >> $LOGFILE
1937     fi
1938     if [ -n "${NIGHTLY_PARENT_TOOLS_ROOT}" ]; then
1939         echo "\n=== Copying tools proto area to $NIGHTLY_PARENT_TOOLS_R
1940         tee -a $LOGFILE >> $mail_msg_file
1941         rm -rf $NIGHTLY_PARENT_TOOLS_ROOT/*
1942         mkdir -p $NIGHTLY_PARENT_TOOLS_ROOT
1943         if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
1944             ( cd $TOOLS_PROTO; tar cf - . |
1945             ( cd $NIGHTLY_PARENT_TOOLS_ROOT;
1946             umask 0; tar xpf - ) ) 2>&1 |
1947             tee -a $mail_msg_file >> $LOGFILE
1948         fi
1949     fi
1950 fi

1952 #
1953 # ELF verification: ABI (-A) and runtime (-r) checks
1954 #
1955 if [[ ($build_ok = y) && (($A_FLAG = y) || ($r_FLAG = y)) ]]; then
1956     # Directory ELF-data.$MACH holds the files produced by these tests.
1957     elf_ddir=$SRC/ELF-data.$MACH

1959     # If there is a previous ELF-data backup directory, remove it. Then,
1960     # rotate current ELF-data directory into its place and create a new
1961     # empty directory
1962     rm -rf $elf_ddir.ref
1963     if [[ -d $elf_ddir ]]; then
1964         mv $elf_ddir $elf_ddir.ref
1965     fi
1966     mkdir -p $elf_ddir

1968     # Call find_elf to produce a list of the ELF objects in the proto area.
1969     # This list is passed to check_rtime and interface_check, preventing
1970     # them from separately calling find_elf to do the same work twice.
1971     find_elf -fr $checkroot > $elf_ddir/object_list

1973     if [[ $A_FLAG = y ]]; then
1974         echo "\n=== Check versioning and ABI information ===\n" | \
1975         tee -a $LOGFILE >> $mail_msg_file

1977     # Produce interface description for the proto. Report errors.
1978     interface_check -o -w $elf_ddir -f object_list \
1979     -i interface -E interface.err
1980     if [[ -s $elf_ddir/interface.err ]]; then
1981         tee -a $LOGFILE < $elf_ddir/interface.err \
1982         >> $mail_msg_file
1983         build_extras_ok=n
1984     fi

1986     # If ELF_DATA_BASELINE_DIR is defined, compare the new interface
1987     # description file to that from the baseline gate. Issue a
1988     # warning if the baseline is not present, and keep going.
1989     if [[ "$ELF_DATA_BASELINE_DIR" != '' ]]; then
1990         base_ifile="$ELF_DATA_BASELINE_DIR/interface"

1992         echo "\n=== Compare versioning and ABI information" \
1993         "to baseline ===\n" | \
1994         tee -a $LOGFILE >> $mail_msg_file
1995         echo "Baseline: $base_ifile\n" >> $LOGFILE

1997         if [[ -f $base_ifile ]]; then
1998             interface_cmp -d -o $base_ifile \

```

```

1999         $elf_ddir/interface > $elf_ddir/interface.cm
2000     if [[ -s $elf_ddir/interface.cmp ]]; then
2001         echo | tee -a $LOGFILE >> $mail_msg_file
2002         tee -a $LOGFILE < \
2003             $elf_ddir/interface.cmp \
2004             >> $mail_msg_file
2005         build_extras_ok=n
2006     fi
2007     else
2008         echo "baseline not available. comparison" \
2009         "skipped" | \
2010         tee -a $LOGFILE >> $mail_msg_file
2011     fi

2013     fi
2014 fi

2016     if [[ $r_FLAG = y ]]; then
2017         echo "\n=== Check ELF runtime attributes ===\n" | \
2018         tee -a $LOGFILE >> $mail_msg_file

2020     # If we're doing a DEBUG build the proto area will be left
2021     # with debuggable objects, thus don't assert -s.
2022     if [[ $D_FLAG = y ]]; then
2023         rtime_sflag=""
2024     else
2025         rtime_sflag="-s"
2026     fi
2027     check_rtime -i -m -v $rtime_sflag -o -w $elf_ddir \
2028     -D object_list -f object_list -E runtime.err \
2029     -I runtime.attr.raw
2030     if (( $? != 0 )); then
2031         build_extras_ok=n
2032     fi

2034     # check_rtime -I output needs to be sorted in order to
2035     # compare it to that from previous builds.
2036     sort $elf_ddir/runtime.attr.raw > $elf_ddir/runtime.attr
2037     rm $elf_ddir/runtime.attr.raw

2039     # Report errors
2040     if [[ -s $elf_ddir/runtime.err ]]; then
2041         tee -a $LOGFILE < $elf_ddir/runtime.err \
2042         >> $mail_msg_file
2043         build_extras_ok=n
2044     fi

2046     # If there is an ELF-data directory from a previous build,
2047     # then diff the attr files. These files contain information
2048     # about dependencies, versioning, and runpaths. There is some
2049     # overlap with the ABI checking done above, but this also
2050     # flushes out non-ABI interface differences along with the
2051     # other information.
2052     echo "\n=== Diff ELF runtime attributes" \
2053     "(since last build) ===\n" | \
2054     tee -a $LOGFILE >> $mail_msg_file >> $mail_msg_file

2056     if [[ -f $elf_ddir.ref/runtime.attr ]]; then
2057         diff $elf_ddir.ref/runtime.attr \
2058         $elf_ddir/runtime.attr \
2059         >> $mail_msg_file
2060     fi
2061 fi

2063     # If -u set, copy contents of ELF-data.$MACH to the parent workspace.
2064     if [[ "$u_FLAG" = "y" ]]; then

```

```

2065     p_elf_ddir=$PARENT_WS/usr/src/ELF-data.$MACH
2067     # If parent lacks the ELF-data.$MACH directory, create it
2068     if [ [ ! -d $p_elf_ddir ] ]; then
2069         staffer mkdir -p $p_elf_ddir
2070     fi
2072     # These files are used asynchronously by other builds for ABI
2073     # verification, as above for the -A option. As such, we require
2074     # the file replacement to be atomic. Copy the data to a temp
2075     # file in the same filesystem and then rename into place.
2076     (
2077         cd $elf_ddir
2078         for elf_dfile in *; do
2079             staffer cp $elf_dfile \
2080                 ${p_elf_ddir}/${elf_dfile}.new
2081             staffer mv -f ${p_elf_ddir}/${elf_dfile}.new \
2082                 ${p_elf_ddir}/${elf_dfile}
2083         done
2084     )
2085 fi
2086 fi
2088 # DEBUG lint of kernel begins
2090 if [ "$i_CMD_LINE_FLAG" = "n" -a "$l_FLAG" = "y" ]; then
2091     if [ "$LINTDIRS" = "" ]; then
2092         # LINTDIRS="$SRC/uts y $SRC/stand y $SRC/psm y"
2093         LINTDIRS="$SRC y"
2094     fi
2095     set $LINTDIRS
2096     while [ $# -gt 0 ]; do
2097         done dolint $1 $2; shift; shift
2098     done
2099 else
2100     echo "\n==== No '$MAKE lint' ==== \n" >> $LOGFILE
2101 fi
2103 # "make check" begins
2105 if [ "$i_CMD_LINE_FLAG" = "n" -a "$C_FLAG" = "y" ]; then
2106     # remove old check.out
2107     rm -f $SRC/check.out
2109     rm -f $SRC/check-${MACH}.out
2110     cd $SRC
2111     $MAKE -ek check ROOT="$checkroot" 2>&1 | tee -a $SRC/check-${MACH}.out \
2112         >> $LOGFILE
2113     echo "\n==== cstyle/hdrchk errors ==== \n" >> $mail_msg_file
2115     grep ":" $SRC/check-${MACH}.out |
2116     egrep -v "Ignoring unknown host" | \
2117     sort | uniq | tee $TMPDIR/check_errors >> $mail_msg_file
2119     if [ [ -s $TMPDIR/check_errors ] ]; then
2120         build_extras_ok=n
2121     fi
2122 else
2123     echo "\n==== No '$MAKE check' ==== \n" >> $LOGFILE
2124 fi
2126 echo "\n==== Find core files ==== \n" | \
2127 tee -a $LOGFILE >> $mail_msg_file
2129 find $abssrkdirs -name core -a -type f -exec file {} \; | \
2130 tee -a $LOGFILE >> $mail_msg_file

```

```

2132 if [ "$f_FLAG" = "y" -a "$build_ok" = "y" ]; then
2133     echo "\n==== Diff unreferenced files (since last build) ==== \n" \
2134         | tee -a $LOGFILE >> $mail_msg_file
2135     rm -f $SRC/unref-${MACH}.ref
2136     if [ -f $SRC/unref-${MACH}.out ]; then
2137         mv $SRC/unref-${MACH}.out $SRC/unref-${MACH}.ref
2138     fi
2140     findunref -S $SCM_TYPE -t $SRC/.build.timestamp -s usr $CODEMGR_WS \
2141         ${TOOLS}/findunref/exception_list 2>> $mail_msg_file | \
2142         sort > $SRC/unref-${MACH}.out
2144     if [ [ ! -f $SRC/unref-${MACH}.ref ] ]; then
2145         cp $SRC/unref-${MACH}.out $SRC/unref-${MACH}.ref
2146     fi
2148     diff $SRC/unref-${MACH}.ref $SRC/unref-${MACH}.out >> $mail_msg_file
2149 fi
2151 # Verify that the usual lists of files, such as exception lists,
2152 # contain only valid references to files. If the build has failed,
2153 # then don't check the proto area.
2154 CHECK_PATHS=${CHECK_PATHS:-y}
2155 if [ "$CHECK_PATHS" = y -a "$N_FLAG" != y ]; then
2156     echo "\n==== Check lists of files ==== \n" | tee -a $LOGFILE \
2157         >> $mail_msg_file
2158     arg=-b
2159     [ "$build_ok" = y ] && arg=
2160     checkpaths $arg $checkroot > $SRC/checkpaths.out 2>&1
2161     if [ [ -s $SRC/checkpaths.out ] ]; then
2162         tee -a $LOGFILE < $SRC/checkpaths.out >> $mail_msg_file
2163         build_extras_ok=n
2164     fi
2165 fi
2167 if [ "$M_FLAG" != "y" -a "$build_ok" = y ]; then
2168     echo "\n==== Impact on file permissions ==== \n" \
2169         >> $mail_msg_file
2171     abspkg=
2172     for d in $abssrkdirs; do
2173         if [ -d "$d/pkg" ]; then
2174             abspkg="$abspkg $d"
2175         fi
2176     done
2178     if [ -n "$abspkg" ]; then
2179         for d in "$abspkg"; do
2180             ( cd $d/pkg ; $MAKE -e pmodes ) >> $mail_msg_file
2181         done
2182     fi
2183 fi
2185 if [ "$w_FLAG" = "y" -a "$build_ok" = "y" ]; then
2186     if [ [ "$MULTI_PROTO" = no || "$D_FLAG" = y ] ]; then
2187         do_wsdiff DEBUG $ROOT.prev $ROOT
2188     fi
2190     if [ [ "$MULTI_PROTO" = yes && "$F_FLAG" = n ] ]; then
2191         do_wsdiff non-DEBUG $ROOT.nd.prev $ROOT-nd
2192     fi
2193 fi
2195 END_DATE=`date`
2196 echo "==== Nightly $maketype build completed: $END_DATE ==== " | \

```

```
2197 tee -a $LOGFILE >> $build_time_file

2199 typeset -i10 hours
2200 typeset -Z2 minutes
2201 typeset -Z2 seconds

2203 elapsed_time=$SECONDS
2204 ((hours = elapsed_time / 3600 ))
2205 ((minutes = elapsed_time / 60 % 60))
2206 ((seconds = elapsed_time % 60))

2208 echo "\n==== Total build time ==== | \
2209 tee -a $LOGFILE >> $build_time_file
2210 echo "\nreal    ${hours}:${minutes}:${seconds}" | \
2211 tee -a $LOGFILE >> $build_time_file

2213 if [ "$u_FLAG" = "y" -a "$f_FLAG" = "y" -a "$build_ok" = "y" ]; then
2214     staffer cp ${SRC}/unref-${MACH}.out $PARENT_WS/usr/src/

2216     #
2217     # Produce a master list of unreferenced files -- ideally, we'd
2218     # generate the master just once after all of the nightlies
2219     # have finished, but there's no simple way to know when that
2220     # will be. Instead, we assume that we're the last nightly to
2221     # finish and merge all of the unref-${MACH}.out files in
2222     # $PARENT_WS/usr/src/. If we are in fact the final ${MACH} to
2223     # finish, then this file will be the authoritative master
2224     # list. Otherwise, another ${MACH}'s nightly will eventually
2225     # overwrite ours with its own master, but in the meantime our
2226     # temporary "master" will be no worse than any older master
2227     # which was already on the parent.
2228     #

2230     set -- $PARENT_WS/usr/src/unref-*.out
2231     cp "$1" ${TMPDIR}/unref.merge
2232     shift

2234     for unreffile; do
2235         comm -12 ${TMPDIR}/unref.merge "$unreffile" > ${TMPDIR}/unref.$$
2236         mv ${TMPDIR}/unref.$$ ${TMPDIR}/unref.merge
2237     done

2239     staffer cp ${TMPDIR}/unref.merge $PARENT_WS/usr/src/unrefmaster.out
2240 fi

2242 #
2243 # All done save for the sweeping up.
2244 # (whichever exit we hit here will trigger the "cleanup" trap which
2245 # optionally sends mail on completion).
2246 #
2247 if [[ "$build_ok" == "y" ]]; then
2248     if [[ "$W_FLAG" == "y" || "$build_extras_ok" == "y" ]]; then
2249         exit 0
2250     fi
2251 fi

2253 exit 1
```