```
**********************************************************
    7213 Thu Aug 15 11:59:46 2013
new/usr/src/Makefile
4028 remove CLOSED_IS_PRESENT
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012 by Delphix. All rights reserved.
#

#
# Makefile for system source
#
# include global definitions
include Makefile.master
#
# the Targetdirs file is the AT&T target.dirs file in a makefile format.
# it defines TARGETDIRS and ROOTDIRS.
include Targetdirs

COMMON_SUBDIRS= uts lib cmd ucblib ucbcmd psm man test
sparc_SUBDIRS= stand
i386_SUBDIRS= grub

#
# sparc needs to build stand before psm
#
$(SPARC_BLD)psm: stand

SUBDIRS= $(COMMON_SUBDIRS) $($(MACH)_SUBDIRS)

HDRSUBDIRS=     uts head lib cmd

# UCB headers are bug-for-bug compatible and not checkable against the header
# standards.
#
CHKHDRSUBDIRS=  head uts lib

#
# Headers that can be built in parallel
#
PARALLEL_HEADERS = sysheaders userheaders libheaders cmdheaders

#
# Directories that can be built in parallel
```

```
#
PARALLEL_DIRS = uts lib man

# The check target also causes smf(5) service manifests to be validated.
CHKMFSTSUBDIRS= cmd

MSGSUBDIRS=     cmd ucbcmd lib
DOMAINS= \
        SUNW_OST_ADMIN \
        SUNW_OST_NETRPC \
        SUNW_OST_OSCMD \
        SUNW_OST_OSLIB \
        SUNW_OST_UCBCMD \
        SUNW_OST_ZONEINFO

MSGDDIRS=       $(DOMAINS:%=$(MSGROOT)/%)
MSGDIRS=        $(MSGROOT) $(MSGDDIRS) $(MSGROOT)/LC_TIME

all :=                  TARGET= all
install :=              TARGET= install
install1 :=             TARGET= install
install2 :=             TARGET= install
install_h :=            TARGET= install_h
clean :=                TARGET= clean
clobber :=              TARGET= clobber
check :=                TARGET= check

.KEEP_STATE:

#
# Note: install does not cause a build in pkg.  To build packages,
#       cd pkg and do a 'make install'
#

all: mapfiles closedbins sgs .WAIT $(SUBDIRS) pkg

#
# The _msg build is a two-step process.  First, the _msg dependency
# causes recursive makes in $(MSGSUBDIRS), which stages raw message
# files in $(ROOT)/catalog.  Second, the action from the install
# target rule causes those messages to be post-processed from where
# they were staged in $(ROOT)/catalog, and the results placed into the
# proto area.
#
# The stage-licenses target causes the license files needed for
# packaging to be pulled from $(SRC) and $(CLOSED) and staged in
# $(ROOT)/licenses.
#
install: install1 install2 _msg stage-licenses
        @cd msg; pwd; $(MAKE) _msg
        @rm -rf "$(ROOT)/catalog"

stage-licenses: install2
        @cd pkg; pwd; $(MAKE) stage-licenses

install1: mapfiles closedbins sgs

install2: install1 $(SUBDIRS)

_msg: _msgdirs rootdirs install2 FRC
        @for m in $(MSGSUBDIRS); do \
                cd $$m; pwd; $(MAKE) _msg; cd ..; \
        done

mapfiles: bldtools
        @cd common/mapfiles; pwd; $(MAKE) install
```

```
clean clobber: $(SUBDIRS) head pkg

closedbins: bldtools $(ROOTDIRS) FRC
        @CLOSED_ROOT="$$ON_CLOSED_BINS/root_$(MACH)$${RELEASE_BUILD+-nd}"; \
        if [ "$$CLOSED_IS_PRESENT" = no ]; then \
        if [ ! -d "$$CLOSED_ROOT" ]; then \
                $(ECHO) "Error: ON_CLOSED_BINS must point to closed" \
                    "binaries."; \
                $(ECHO) "root_$(MACH)$${RELEASE_BUILD+-nd} is not" \
                    "present in $$ON_CLOSED_BINS."; \
                exit 1; \
        fi; \
        $(ECHO) "Copying closed binaries from $$CLOSED_ROOT"; \
        (cd $$CLOSED_ROOT; \
            $(TAR) cfX - $(CODEMGR_WS)/exception_lists/closed-bins .) | \
            (cd $(ROOT); $(TAR) xBpf -); \
        ( cd $(ROOT); $(CTFSTRIP) $$(cd $$CLOSED_ROOT; $(FIND) \
            ./kernel ./usr/kernel ./platform/*/kernel -type f -a -perm -u+x | \
            $(EGREP) -vf $(CODEMGR_WS)/exception_lists/closed-bins) )
                    $(EGREP) -vf $(CODEMGR_WS)/exception_lists/closed-bins) ); \
        fi

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

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

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

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

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

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

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

_msgdirs:       $(MSGDIRS)

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

userheaders: FRC
```

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

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

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

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

check:  $(CHKHDRSUBDIRS) $(CHKMFSTSUBDIRS)

#
# Cross-reference customization: skip all of the subdirectories that
# don’t contain actual source code.
#
XRPRUNE = pkg prototypes
XRINCDIRS = uts/common head ucbhead

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

FRC:

#
# Targets for reporting compiler versions; nightly uses these.
#

cc-version:
        @if $($(MACH)_CC) -_versions >/dev/null 2>/dev/null; then \
                $(ECHO) 32-bit compiler;                        \
                $(ECHO) $($(MACH)_CC);                          \
                $($(MACH)_CC) -_versions 2>&1 |                 \
                    $(EGREP) ’^(cw|cc|gcc|primary|shadow)’;     \
        else                                                    \
                __COMPILER=‘$($(MACH)_CC) -_compiler 2>/dev/null || $(TRUE)‘;\
                if [ -z "$$__COMPILER" ]; then                  \
                        $(ECHO) No 32-bit compiler found;       \
                        exit 1;                                 \
                else                                            \
                        $(ECHO) 32-bit compiler;                \
                        $(ECHO) $($(MACH)_CC);                  \
                        $(ECHO) $$__COMPILER;                   \
                        $($(MACH)_CC) -V 2>&1 | head -1;        \
                fi;                                             \
        fi

cc64-version:
        @if $($(MACH64)_CC) -_versions >/dev/null 2>/dev/null; then \
                $(ECHO) 64-bit compiler;                        \
                $(ECHO) $($(MACH64)_CC);                        \
                $($(MACH64)_CC) -_versions 2>&1 |               \
                    $(EGREP) ’^(cw|cc|gcc|primary|shadow)’;     \
        else                                                    \
                __COMPILER=‘$($(MACH64)_CC) -_compiler 2>/dev/null || $(TRUE)‘;\
                if [ -z "$$__COMPILER" ]; then                  \
                        $(ECHO) No 64-bit compiler found;       \
                        exit 1;                                 \
                else                                            \
                        $(ECHO) 64-bit compiler;                \
                        $(ECHO) $($(MACH64)_CC);                \
                        $(ECHO) $$__COMPILER;                   \
                        $($(MACH64)_CC) -V 2>&1 | head -1;      \
                fi;                                             \
	fi

java-version:
	@if [ -x "$(JAVAC)" ]; then			\
		$(ECHO) $(JAVAC);			\
		$(JAVA_ROOT)/bin/java -fullversion 2>&1 | head -1;	\
	else						\
		$(ECHO) No Java compiler found;		\
		exit 1;					\
	fi
```

```
**********************************************************
    8013 Thu Aug 15 11:59:46 2013
new/usr/src/tools/env/developer.sh
4028 remove CLOSED_IS_PRESENT
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
#

#       Configuration variables for the runtime environment of the nightly
# build script and other tools for construction and packaging of releases.
# This script is sourced by 'nightly' and 'bldenv' to set up the environment
# for the build. This example is suitable for building a developers workspace,
# which will contain the resulting packages and archives. It is based off
# the onnv release. It sets NIGHTLY_OPTIONS to make nightly do:
#       check ELF ABI/versioning (-A)
#       runs 'make check' (-C)
#       DEBUG and non-DEBUG builds (-D)
#       runs lint in usr/src (-l plus the LINTDIRS variable)
#       sends mail on completion (-m and the MAILTO variable)
#       creates packages for PIT/RE (-p)
#       checks for changes in ELF runpaths (-r)
#
NIGHTLY_OPTIONS="-ACDlmpr";             export NIGHTLY_OPTIONS

# This is a variable for the rest of the script - GATE doesn't matter to
# nightly itself
GATE=onnv-bugfixes;                     export GATE

# CODEMGR_WS - where is your workspace at (or what should nightly name it)
CODEMGR_WS="/builds/$GATE";                     export CODEMGR_WS

# PARENT_WS is used to determine the parent of this workspace. This is
# for the options that deal with the parent workspace (such as where the
# proto area will go).
#
# If you use this, it must be local (or nfs): nightly cannot copy
# over ssh or http.
PARENT_WS="/ws/onnv-gate";                      export PARENT_WS

# CLONE_WS is the workspace nightly should do a bringover from.
CLONE_WS="ssh://anonhg@onnv.sfbay.sun.com//export/onnv-clone";  export CLONE_WS

# CLOSED_CLONE_WS is the workspace from which nightly should acquire
# the usr/closed tree.
```

```
CLOSED_CLONE_WS="${CLONE_WS}/usr/closed";       export CLOSED_CLONE_WS

# This flag controls whether to build the closed source.  If
# undefined, nightly(1) and bldenv(1) will set it according to whether
# the closed source tree is present.  CLOSED_IS_PRESENT="no" means not
# building the closed sources.
# CLOSED_IS_PRESENT="yes";              export CLOSED_IS_PRESENT

# The bringover, if any, is done as STAFFER.
# Set STAFFER to your own login as gatekeeper or developer
# The point is to use group "staff" and avoid referencing the parent
# workspace as root.
# Some scripts optionally send mail messages to MAILTO.
#
STAFFER=nobody;                         export STAFFER
MAILTO=$STAFFER;                        export MAILTO

# The project (see project(4)) under which to run this build.  If not
# specified, the build is simply run in a new task in the current project.
BUILD_PROJECT=;                         export BUILD_PROJECT

# You should not need to change the next four lines
LOCKNAME="`basename $CODEMGR_WS`_nightly.lock"; export LOCKNAME
ATLOG="$CODEMGR_WS/log";                        export ATLOG
LOGFILE="$ATLOG/nightly.log";                   export LOGFILE
MACH=`uname -p`;                                export MACH

# When the -A flag is specified, and ELF_DATA_BASELINE_DIR is defined,
# the ELF interface description file resulting from the build is compared
# to that from the specified directory. This ensures that our object
# versioning evolves in a backward compatible manner.
#
# You should not need to change this unless you wish to use locally cached
# baseline files. If you use this, it must be local (or nfs): nightly cannot
# copy over ssh or http.
#
ELF_DATA_BASELINE_DIR="/ws/onnv-gate/usr/src/ELF-data-baseline.$MACH";  export E

# This is usually just needed if the closed tree is missing, or when
# building a project gate with the -O (cap oh) flag.
# ON_CRYPTO_BINS="$PARENT_WS/packages/$MACH/on-crypto.$MACH.tar.bz2"
# export ON_CRYPTO_BINS

# REF_PROTO_LIST - for comparing the list of stuff in your proto area
# with. Generally this should be left alone, since you want to see differences
# from your parent (the gate).
#
REF_PROTO_LIST=$PARENT_WS/usr/src/proto_list_${MACH}; export REF_PROTO_LIST

#
#       build environment variables, including version info for mcs, motd,
# motd, uname and boot messages. Mostly you shouldn't change this except
# when the release slips (nah) or you move an environment file to a new
# release
#
ROOT="$CODEMGR_WS/proto/root_${MACH}";  export ROOT
SRC="$CODEMGR_WS/usr/src";              export SRC
VERSION="$GATE";                        export VERSION

#
# the RELEASE and RELEASE_DATE variables are set in Makefile.master;
# there might be special reasons to override them here, but that
# should not be the case in general
#
# RELEASE="5.10.1";                     export RELEASE
# RELEASE_DATE="October 2007";          export RELEASE_DATE
# proto area in parent for optionally depositing a copy of headers and
# libraries corresponding to the protolibs target
# not applicable given the NIGHTLY_OPTIONS
#
PARENT_ROOT=$PARENT_WS/proto/root_$MACH; export PARENT_ROOT
PARENT_TOOLS_ROOT=$PARENT_WS/usr/src/tools/proto/root_$MACH-nd; export PARENT_TO

#
# Package creation variables.  You probably shouldn't change these,
# either.
#
# PKGARCHIVE determines where repositories will be created.
#
# PKGPUBLISHER* control the publisher settings for those repositories.
#
PKGARCHIVE="${CODEMGR_WS}/packages/${MACH}/nightly";    export PKGARCHIVE
# PKGPUBLISHER_REDIST="on-redist";                      export PKGPUBLISHER_REDI
# PKGPUBLISHER_NONREDIST="on-extra";                    export PKGPUBLISHER_NONR

# we want make to do as much as it can, just in case there's more than
# one problem.
MAKEFLAGS=k;    export MAKEFLAGS

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

# Build tools - don't set these unless you know what you're doing.  These
# variables allows you to get the compilers and onbld files locally or
# through cachefs.  Set BUILD_TOOLS to pull everything from one location.
# Alternately, you can set ONBLD_TOOLS to where you keep the contents of
# SUNWonbld and SPRO_ROOT to where you keep the compilers.
#
#BUILD_TOOLS=/opt;                              export BUILD_TOOLS
#ONBLD_TOOLS=/opt/onbld;                        export ONBLD_TOOLS
#SPRO_ROOT=/opt/SUNWspro;                       export SPRO_ROOT

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

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

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

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

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

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

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

# POST_NIGHTLY can be any command to be run at the end of nightly.  See
# nightly(1) for interactions between environment variables and this command.
#POST_NIGHTLY=
**********************************************************
    8626 Thu Aug 15 11:59:46 2013
new/usr/src/tools/env/gatekeeper.sh
4028 remove CLOSED_IS_PRESENT
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
#

#       Configuration variables for the runtime environment of the nightly
# build script and other tools for construction and packaging of releases.
# This script is sourced by 'nightly' and 'bldenv' to set up the environment
# for the build. This example is suitable for building a gate,
# which will contain the resulting packages and archives (builds of the gate
# are done in children and then the resulting archives, packages, and proto
# area are put into the parent for everyone to use). It is based off
# the onnv release. It sets NIGHTLY_OPTIONS to make nightly do:
#       DEBUG and non-DEBUG builds (-D)
#       creates packages for PIT/RE (-p)
#       checks for new interfaces in libraries (-A)
#       runs 'make check' (-C)
#       runs lint in usr/src (-l plus the LINTDIRS variable)
#       sends mail on completion (-m and the MAILTO variable)
#       updates the protolist in the parent for children to compare with (-u)
#       updates the proto area in the parent when done (-U)
#       checks for changes in ELF runpaths (-r)
#       checks for changes in unreferenced files (-f)
#
NIGHTLY_OPTIONS="-ADClmpuUrf";          export NIGHTLY_OPTIONS

# This is a variable for the rest of the script - GATE doesn't matter to
# nightly itself
GATE=onnv-gate;                                 export GATE

# CODEMGR_WS - where is your workspace at (or what should nightly name it)
# there is only one definition here, which assumes all the gate build machines
# (sparc and x86) are set up the same. But remember, this is a script, so
# you _could_ look at $MACH or 'uname -n' and set these variables differently.
CODEMGR_WS="/builds/$GATE";                     export CODEMGR_WS

# PARENT_WS is used to determine the parent of this workspace. This is
# for the options that deal with the parent workspace (such as where the
# proto area will go).
#
# If you use this, it must be local (or nfs): nightly cannot copy
```

```
# over ssh or http.
PARENT_WS="/ws/$GATE";                          export PARENT_WS

# CLONE_WS is the workspace nightly should do a bringover from.
CLONE_WS="ssh://anonhg@onnv.sfbay.sun.com//export/onnv-clone";  export CLONE_WS

# CLOSED_CLONE_WS is the workspace from which nightly will acquire the
# usr/closed tree.
CLOSED_CLONE_WS="${CLONE_WS}/usr/closed"
export CLOSED_CLONE_WS

# This flag controls whether to build the closed source.  If
# undefined, nightly(1) and bldenv(1) will set it according to whether
# the closed source tree is present.  CLOSED_IS_PRESENT="no" means not
# building the closed sources.
# CLOSED_IS_PRESENT="yes";              export CLOSED_IS_PRESENT

# The bringover, if any, is done as STAFFER.
# Set STAFFER to your own login as gatekeeper or integration engineer.
# The point is to use group "staff" and avoid referencing the parent
# workspace as root.
# Some scripts optionally send mail messages to MAILTO.
#
STAFFER=nobody;                         export STAFFER
MAILTO=$STAFFER;                        export MAILTO

# The project (see project(4)) under which to run this build.  If not
# specified, the build is simply run in a new task in the current project.
BUILD_PROJECT=;                         export BUILD_PROJECT

# You should not need to change the next four lines
LOCKNAME="'basename $CODEMGR_WS'_nightly.lock"; export LOCKNAME
ATLOG="$CODEMGR_WS/log";                        export ATLOG
LOGFILE="$ATLOG/nightly.log";                   export LOGFILE
MACH='uname -p';                                export MACH

# When the -A flag is specified, and ELF_DATA_BASELINE_DIR is defined,
# the ELF interface description file resulting from the build is compared
# to that from the specified directory. This ensures that our object
# versioning evolves in a backward compatible manner.
#
# You should not need to change this unless you wish to use locally cached
# baseline files. If you use this, it must be local (or nfs): nightly cannot
# copy over ssh or http.
#
ELF_DATA_BASELINE_DIR="/ws/onnv-gate/usr/src/ELF-data-baseline.$MACH";  export E

# This is usually just needed if the closed tree is missing, or when
# building a project gate with the -O (cap oh) flag.
# ON_CRYPTO_BINS="$PARENT_WS/packages/$MACH/on-crypto.$MACH.tar.bz2"
# export ON_CRYPTO_BINS

# REF_PROTO_LIST - for comparing the list of stuff in your proto area
# with. Generally this should be left alone, since you want to see differences
# between todays build and yesterdays.
#
REF_PROTO_LIST=$PARENT_WS/usr/src/proto_list_${MACH}; export REF_PROTO_LIST

#
#       build environment variables, including version info for mcs, motd,
# motd, uname and boot messages. Mostly you shouldn't change this except
# when the release slips (nah) or when starting a new release.
#
ROOT="$CODEMGR_WS/proto/root_${MACH}";  export ROOT
SRC="$CODEMGR_WS/usr/src";              export SRC
VERSION="$GATE";                        export VERSION
```

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

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

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


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

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

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

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

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

#
# Reference to binary-only IA32 IHV packages
#
#IA32_IHV_BINARY_PKGS=/ws/${GATE}-ihv-bin
#export IA32_IHV_BINARY_PKGS
```

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

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

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

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

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

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

```
function copy_kmdb {
        typeset kmdbtgtdir=$INSTALL_FILES/platform/$KARCH/$GLOMNAME/misc
        typeset bitdirs=
        typeset isadir=
        typeset b64srcdir=
        typeset b64tgtdir=
        typeset b32srcdir=
        typeset b32tgtdir=
        typeset machdir=
        typeset platdir=

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

        if [[ $(mach) = "i386" ]] ; then
                isadir="intel"
                b64srcdir="amd64"
                b64tgtdir="amd64"
                b32srcdir="ia32"
                b32tgtdir="."
        else
                isadir="sparc"
                b64srcdir="v9"
                b64tgtdir="sparcv9"
                b32srcdir="v7"
                b32tgtdir="."
        fi

        typeset foundkmdb=no
        typeset kmdbpath=
        typeset destdir=

        platdir=$INSTALL_FILES/platform/$KARCH/$GLOMNAME
        if [[ $GLOM = "yes" ]] ; then
                machdir=$platdir
        else
                machdir=$INSTALL_FILES/kernel
        fi

        srctrees=$SRC
        if [[ -d $SRC/../closed && "$CLOSED_IS_PRESENT" != no ]]; then
                srctrees="$srctrees $SRC/../closed"
        else
        if [ -z "$ON_CRYPTO_BINS" ]; then
                echo "Warning: ON_CRYPTO_BINS not set; pre-signed" \
                    "crypto not provided."
        fi
        fi
        if [[ $WANT64 = "yes" ]] ; then
                # kmdbmod for sparc and x86 are built and installed
                # in different places
                if [[ $(mach) = "i386" ]] ; then
                        kmdbpath=$SRC/cmd/mdb/$isadir/$b64srcdir/kmdb/kmdbmod
                        destdir=$machdir/misc/$b64tgtdir
```

```
                else
                        kmdbpath=$SRC/cmd/mdb/$KARCH/$b64srcdir/kmdb/kmdbmod
                        destdir=$platdir/misc/$b64tgtdir
                fi

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

                        for tree in $srctrees; do
                                kmdb_copy_machkmods \
                                    $tree/cmd/mdb/$isadir/$b64srcdir \
                                    $machdir/kmdb/$b64tgtdir
                                kmdb_copy_karchkmods $tree/cmd/mdb/$KARCH \
                                    $platdir/kmdb/$b64tgtdir $b64srcdir
                        done
                fi
        fi

        if [[ $WANT32 = "yes" ]] ; then
                kmdbpath=$SRC/cmd/mdb/$isadir/$b32srcdir/kmdb/kmdbmod
                destdir=$machdir/misc/$b32tgtdir

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

                        for tree in $srctrees; do
                                kmdb_copy_machkmods \
                                    $tree/cmd/mdb/$isadir/$b32srcdir \
                                    $machdir/kmdb/$b32tgtdir
                                kmdb_copy_karchkmods $tree/cmd/mdb/$KARCH \
                                    $platdir/kmdb/$b32tgtdir $b32srcdir
                        done
                fi
        fi

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

        STATE=2
        save_state
        return
}
```
______***unchanged_portion_omitted_***

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

# main
builtin basename

# boolean flags (true/false)
typeset flags=(
        typeset c=false
        typeset f=false
        typeset d=false
        typeset O=false
        typeset o=false
        typeset t=true
        typeset s=(
                typeset e=false
                typeset h=false
                typeset d=false
                typeset o=false
        )
)

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

OPTIND=1
SUFFIX="-nd"

while getopts -a "${progname}" "${USAGE}" OPT ; do
    case ${OPT} in
          c)    flags.c=true  ;;
          +c)   flags.c=false ;;
          f)    flags.f=true  ;;
          +f)   flags.f=false ;;
          d)    flags.d=true  SUFFIX=""    ;;
          +d)   flags.d=false SUFFIX="-nd" ;;
          t)    flags.t=true  ;;
          +t)   flags.t=false ;;
          S)    set_S_flag "$OPTARG" ;;
          \?)   usage ;;
    esac
done
shift $((OPTIND-1))

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

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

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

```
        LD_LIBRARY_PATH \
        LD_AUDIT \
        LD_BIND_NOW \
        LD_BREADTH \
        LD_CONFIG \
        LD_DEBUG \
        LD_FLAGS \
        LD_LIBRARY_PATH_64 \
        LD_NOVERSION \
        LD_ORIGIN \
        LD_LOADFLTR \
        LD_NOAUXFLTR \
        LD_NOCONFIG \
        LD_NODIRCONFIG \
        LD_NOOBJALTER \
        LD_PRELOAD \
        LD_PROFILE \
        CONFIG \
        GROUP \
        OWNER \
        REMOTE \
        ENV \
        ARCH \
        CLASSPATH

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

if [[ -f "$1" ]]; then
        if [[ "$1" == */* ]]; then
                source "$1"
        else
                source "./$1"
        fi
else
        if [[ -f "/opt/onbld/env/$1" ]]; then
                source "/opt/onbld/env/$1"
        else
                printf \
                    'Cannot find env file as either %s or /opt/onbld/env/%s\n' \
                    "$1" "$1"
                exit 1
        fi
fi
shift

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

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

# must match the getopts in nightly.sh
OPTIND=1
NIGHTLY_OPTIONS="-${NIGHTLY_OPTIONS#-}"
while getopts '+0AaBCDdFfGIilMmNnOopRrS:tUuWwXxz' FLAG "$NIGHTLY_OPTIONS"
do
        case "$FLAG" in
          O)    flags.O=true  ;;
          +O)   flags.O=false ;;
          o)    flags.o=true  ;;
          +o)   flags.o=false ;;
          t)    flags.t=true  ;;
          +t)   flags.t=false ;;
          S)    set_S_flag "$OPTARG" ;;
          *)    ;;
        esac
done

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

export INTERNAL_RELEASE_BUILD=

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

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

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

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

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

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

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

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

```
fi

if [[ -z "$CLOSED_IS_PRESENT" ]]; then
        if [[ -d $SRC/../closed ]]; then
                export CLOSED_IS_PRESENT="yes"
        else
                export CLOSED_IS_PRESENT="no"
        fi
fi

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

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

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

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

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

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

export DMAKE_MODE=${DMAKE_MODE:-parallel}

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

TMPDIR="/tmp"

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

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

# a la ws
ENVLDLIBS1=
ENVLDLIBS2=
ENVLDLIBS3=
ENVCPPFLAGS1=
ENVCPPFLAGS2=
ENVCPPFLAGS3=
ENVCPPFLAGS4=
PARENT_ROOT=
PARENT_TOOLS_ROOT=
if [[ "$MULTI_PROTO" != "yes" && "$MULTI_PROTO" != "no" ]]; then
        printf \
            'WARNING: invalid value for MULTI_PROTO (%s); setting to "no".\n' \
            "$MULTI_PROTO"
        export MULTI_PROTO="no"
fi

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

export TONICBUILD="#"

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

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

export \
        ENVLDLIBS1 \
        ENVLDLIBS2 \
        ENVLDLIBS3 \
        ENVCPPFLAGS1 \
        ENVCPPFLAGS2 \
        ENVCPPFLAGS3 \
        ENVCPPFLAGS4 \
        MAKEFLAGS \
        PARENT_ROOT \
        PARENT_TOOLS_ROOT

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

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

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

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

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

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

```
        fi

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

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

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

# not reached
```

****
3889 Thu Aug 15 11:59:47 2013
new/usr/src/tools/scripts/checkpaths.sh
4028 remove CLOSED_IS_PRESENT
****

```
#!/bin/ksh -p
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

# Quis custodiet ipsos custodies?

if [ -z "$SRC" ]; then
        SRC=$CODEMGR_WS/usr/src
fi

if [ -z "$CODEMGR_WS" -o ! -d "$CODEMGR_WS" -o ! -d "$SRC" ]; then
        echo "$0: must be run from within a workspace."
        exit 1
fi

cd $CODEMGR_WS || exit 1

# Use -b to tell this script to ignore derived (built) objects.
if [ "$1" = "-b" ]; then
        b_flg=y
fi

# Not currently used; available for temporary workarounds.
args="-k NEVER_CHECK"

# We intentionally don't depend on $MACH here, and thus no $ROOT.  If
# a proto area exists, then we use it.  This allows this script to be
# run against gates (which should contain both SPARC and x86 proto
# areas), build workspaces (which should contain just one proto area),
# and unbuilt workspaces (which contain no proto areas).
if [ "$b_flg" = y ]; then
        rootlist=
elif [ $# -gt 0 ]; then
        rootlist=$*
else
        rootlist="$CODEMGR_WS/proto/root_sparc $CODEMGR_WS/proto/root_i386"
fi
```

```
# If the closed source is not present, then exclude IKE from validation.
if [ "$CLOSED_IS_PRESENT" = no ]; then
        excl="-e ^usr/include/ike/"
fi

for ROOT in $rootlist
do
        case "$ROOT" in
        *sparc|*sparc-nd)
                arch=sparc
                ;;
        *i386|*i386-nd)
                arch=i386
                ;;
        *)
                echo "$ROOT has unknown architecture." >&2
                exit 1
                ;;
        esac
        if [ -d $ROOT ]; then
                #
                # This is the old-style packaging exception list, from
                # the svr4-specific usr/src/pkgdefs
                #
                [ -f $SRC/pkgdefs/etc/exception_list_$arch ] && \
                        validate_paths '-s/\s*'$arch'$//'  \
                            -e ^usr/include/ike/ -b $ROOT \
                        validate_paths '-s/\s*'$arch'$//' $excl -b $ROOT \
                            $args $SRC/pkgdefs/etc/exception_list_$arch
                #
                # These are the new-style packaging exception lists,
                # from the repository-wide exception_lists/ directory.
                #
                e="$CODEMGR_WS/exception_lists/packaging"
                for f in $e; do
                        if [ -f $f ]; then
                                nawk 'NF == 1 || /[     ]\+'$arch'$/ { print; }'
                                    < $f | validate_paths -b $ROOT -n $f
                        fi
                done
        fi
done

# Two entries in the findunref exception_list deal with things created
# by nightly.  Otherwise, this test could be run on an unmodifed (and
# unbuilt) workspace.  We handle this by flagging the one that is
# present only on a built workspace (./*.out) and the one that's
# present only after a run of findunref (./*.ref) with ISUSED, and
# disabling all checks of them.  The assumption is that the entries
# marked with ISUSED are always known to be good, thus the Latin quote
# at the top of the file.
#
# The exception_list is generated from whichever input files are appropriate
# for this workspace, so checking it obviates the need to check the inputs.

if [ -r $SRC/tools/findunref/exception_list ]; then
        validate_paths -k ISUSED -r -e '^\*' $SRC/tools/findunref/exception_list
fi

if [ -f $SRC/tools/opensolaris/license-list ]; then
        excl=
        if [ "$CLOSED_IS_PRESENT" = no ]; then
                excl="-e ^usr/closed"
        fi
        sed -e 's/$/.descrip/' < $SRC/tools/opensolaris/license-list | \
                validate_paths -n SRC/tools/opensolaris/license-list \
                    -e ^usr/closed
                validate_paths -n SRC/tools/opensolaris/license-list $excl
fi

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

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

exit 0
```

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

#
# Mercurial-specific copy code for copy_source().
# Mercurial-specific copy code for copy_source().  Handles the
# combined open and closed trees.
#
# Returns 0 for success, non-zero for failure.
#
# usage: copy_source_mercurial destdir srcroot
#
function copy_source_mercurial {
        typeset dest=$1
        typeset srcroot=$2
        typeset open_top closed_top

        hg locate -I "$srcroot" | cpio -pd "$dest" >>$LOGFILE 2>&1
        case $srcroot in
        usr)
                open_top=usr
                if [[ "$CLOSED_IS_PRESENT" = yes ]]; then
                        closed_top=usr/closed
                fi
                ;;
        usr/closed*)
                if [[ "$CLOSED_IS_PRESENT" = no ]]; then
                        printf "can't copy %s: closed tree not present.\n" \
                            "$srcroot" | tee -a $mail_msg_file >> $LOGFILE
                        return 1
                fi
                closed_top="$srcroot"
                ;;
        *)
                open_top="$srcroot"
                ;;
        esac

        if [[ -n "$open_top" ]]; then
                hg locate -I "$open_top" | cpio -pd "$dest" >>$LOGFILE 2>&1
        if (( $? != 0 )) ; then
            printf "cpio failed for %s\n" "$dest" |
                tee -a $mail_msg_file >> $LOGFILE
            return 1
        fi
        fi

        if [[ -n "$closed_top" ]]; then
                mkdir -p "$dest/usr/closed" || return 1
                if [[ "$closed_top" = usr/closed ]]; then
                        (cd usr/closed; hg locate |
                            cpio -pd "$dest/usr/closed") >>$LOGFILE 2>&1
                        if (( $? != 0 )) ; then
                            printf "cpio failed for %s/usr/closed\n" \
                                "$dest" | tee -a $mail_msg_file >> $LOGFILE
                            return 1
                        fi
                else
                        # copy subtree of usr/closed
                        closed_top=${closed_top#usr/closed/}
                        (cd usr/closed; hg locate -I "$closed_top" |
                            cpio -pd "$dest/usr/closed") >>$LOGFILE 2>&1
```

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

        return 0
}
______unchanged_portion_omitted_

#
# Verify that the closed tree is present if it needs to be.
#
function check_closed_tree {
        if [[ ! -d "$ON_CLOSED_BINS" ]]; then
                echo "ON_CLOSED_BINS must point to the closed binaries tree."
                echo "If the closed sources are not present," \
                    "ON_CLOSED_BINS"
                echo "must point to the closed binaries tree."
                build_ok=n
                exit 1
        fi
}
______unchanged_portion_omitted_

OPTIND=1
while getopts +inS:tV: FLAG
do
        case $FLAG in
          i )   i_FLAG=y; i_CMD_LINE_FLAG=y
                ;;
          n )   n_FLAG=y
                ;;
          S )
                set_S_flag $OPTARG
                ;;
         +t )   t_FLAG=n
                ;;
          V )   V_FLAG=y
                V_ARG="$OPTARG"
                ;;
         \? )   echo "$USAGE"
                exit 1
                ;;
        esac
done

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

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

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

#
```

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

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

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

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

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

if [ -f $1 ]; then
        if [[ $1 = */* ]]; then
                . $1
        else
                . ./$1
        fi
else
        if [ -f $OPTHOME/onbld/env/$1 ]; then
                . $OPTHOME/onbld/env/$1
        else
                echo "Cannot find env file as either $1 or $OPTHOME/onbld/env/$1
                exit 1
```

```
        fi
fi

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

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

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

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

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

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

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

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

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

```
#
# Note: changes to the option letters here should also be applied to the
#       bldenv script.  'd' is listed for backward compatibility.
#
NIGHTLY_OPTIONS=-${NIGHTLY_OPTIONS#-}
OPTIND=1
while getopts +ABCDdFfGIilMmNnOoPpRrS:TtUuWwXxz FLAG $NIGHTLY_OPTIONS
do
        case $FLAG in
          A )   A_FLAG=y
                #
                # If ELF_DATA_BASELINE_DIR is not defined, and we are on SWAN
                # (based on CLOSED_IS_PRESENT), then refuse to run. The value
                # of ELF version checking is greatly enhanced by including
                # the baseline gate comparison.
                if [ "$CLOSED_IS_PRESENT" = 'yes' -a \
                     "$ELF_DATA_BASELINE_DIR" = '' ]; then
                        echo "ELF_DATA_BASELINE_DIR must be set if the A" \
                            "flag is present in\nNIGHTLY_OPTIONS and closed" \
                            "sources are present. Update environment file."
                        exit 1;
                fi
                ;;
          B )   D_FLAG=y
                ;; # old version of D
          C )   C_FLAG=y
                ;;
          D )   D_FLAG=y
                ;;
          F )   F_FLAG=y
                ;;
          f )   f_FLAG=y
                ;;
          G )   u_FLAG=y
                ;;
          I )   m_FLAG=y
                p_FLAG=y
                u_FLAG=y
                ;;
          i )   i_FLAG=y
                ;;
          l )   l_FLAG=y
                ;;
          M )   M_FLAG=y
                ;;
          m )   m_FLAG=y
                ;;
          N )   N_FLAG=y
                ;;
          n )   n_FLAG=y
                ;;
          O )   O_FLAG=y
                ;;
          o )   o_FLAG=y
                ;;
          P )   P_FLAG=y
                ;; # obsolete
          p )   p_FLAG=y
                ;;
          R )   m_FLAG=y
                p_FLAG=y
                ;;
          r )   r_FLAG=y
                ;;
          S )
                set_S_flag $OPTARG
```

```
                ;;
          T )   T_FLAG=y
                ;; # obsolete
         +t )   t_FLAG=n
                ;;
          U )   if [ -z "${PARENT_ROOT}" ]; then
                        echo "PARENT_ROOT must be set if the U flag is" \
                            "present in NIGHTLY_OPTIONS."
                        exit 1
                fi
                NIGHTLY_PARENT_ROOT=$PARENT_ROOT
                if [ -n "${PARENT_TOOLS_ROOT}" ]; then
                        NIGHTLY_PARENT_TOOLS_ROOT=$PARENT_TOOLS_ROOT
                fi
                U_FLAG=y
                ;;
          u )   u_FLAG=y
                ;;
          W )   W_FLAG=y
                ;;

          w )   w_FLAG=y
                ;;
          X )   # now that we no longer need realmode builds, just
                # copy IHV packages.  only meaningful on x86.
                if [ "$MACH" = "i386" ]; then
                        X_FLAG=y
                fi
                ;;
          x )   XMOD_OPT="-x"
                ;;
         \? )   echo "$USAGE"
                exit 1
                ;;
        esac
done

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

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

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

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

# roots of source trees, both relative to $SRC and absolute.
relsrcdirs="."
abssrcdirs="$SRC"
if [[ -d $CODEMGR_WS/usr/closed && "$CLOSED_IS_PRESENT" != no ]]; then
        relsrcdirs="$relsrcdirs ../closed"
fi
```

```
abssrcdirs=""
for d in $relsrcdirs; do
        abssrcdirs="$abssrcdirs $SRC/$d"
done

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

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

maketype="distributed"
MAKE=dmake
# get the dmake version string alone
DMAKE_VERSION=$( $MAKE -v )
DMAKE_VERSION=${DMAKE_VERSION#*: }
# focus in on just the dotted version number alone
DMAKE_MAJOR=$( echo $DMAKE_VERSION | \
        sed -e 's/.*\<\([^.]*\.[^   ]*\).*$/\1/' )
# extract the second (or final) integer
DMAKE_MINOR=${DMAKE_MAJOR#*.}
DMAKE_MINOR=${DMAKE_MINOR%%.*}
# extract the first integer
DMAKE_MAJOR=${DMAKE_MAJOR%%.*}
CHECK_DMAKE=${CHECK_DMAKE:-y}
# x86 was built on the 12th, sparc on the 13th.
if [ "$CHECK_DMAKE" = "y" -a \
     "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/12" -a \
     "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/13" -a \( \
     "$DMAKE_MAJOR" -lt 7 -o \
     "$DMAKE_MAJOR" -eq 7 -a "$DMAKE_MINOR" -lt 4 \) ]; then
        if [ -z "$DMAKE_VERSION" ]; then
                echo "$MAKE is missing."
                exit 1
        fi
        echo `whence $MAKE`" version is:"
        echo "  ${DMAKE_VERSION}"
        cat <<EOF

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

  CHECK_DMAKE=n
EOF
        exit 1
fi
```

```
export PATH
export MAKE

if [[ "$O_FLAG" = y ]]; then
        export TONICBUILD=""
else
        export TONICBUILD="#"
fi

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

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

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

        export DMAKE_MAX_JOBS=$maxjobs
fi

DMAKE_MODE=parallel;
export DMAKE_MODE

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

#
# if -V flag was given, reset VERSION to V_ARG
#
if [ "$V_FLAG" = "y" ]; then
        VERSION=$V_ARG
fi

#
# Check for IHV root for copying ihv proto area
#
if [ "$X_FLAG" = "y" ]; then
        if [ "$IA32_IHV_ROOT" = "" ]; then
                echo "IA32_IHV_ROOT: must be set for copying ihv proto"
                args_ok=n
        fi
        if [ ! -d "$IA32_IHV_ROOT" ]; then
                echo "$IA32_IHV_ROOT: not found"
                args_ok=n
        fi
        if [ "$IA32_IHV_WS" = "" ]; then
                echo "IA32_IHV_WS: must be set for copying ihv proto"
                args_ok=n
        fi
        if [ ! -d "$IA32_IHV_WS" ]; then
                echo "$IA32_IHV_WS: not found"
                args_ok=n
        fi
fi

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

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

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

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

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

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

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

unset   CFLAGS LD_LIBRARY_PATH LDFLAGS

# create directories that are automatically removed if the nightly script
# fails to start correctly
function newdir {
        dir=$1
        toadd=
        while [ ! -d $dir ]; do
                toadd="$dir $toadd"
                dir=`dirname $dir`
        done
        torm=
        newlist=
        for dir in $toadd; do
                if staffer mkdir $dir; then
```

```
                        newlist="$ISUSER $dir $newlist"
                        torm="$dir $torm"
                else
                        [ -z "$torm" ] || staffer rmdir $torm
                        return 1
                fi
        done
        newdirlist="$newlist $newdirlist"
        return 0
}
______unchanged_portion_omitted_

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

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

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

        typeset -x HGMERGE="/bin/false"

        #
        # If the user has changes, regardless of whether those changes are
        # committed, and regardless of whether those changes conflict, then
        # we'll attempt to merge them either implicitly (uncommitted) or
        # explicitly (committed).
        #
        # These are the messages we'll use to help clarify mercurial output
        # in those cases.
        #
        typeset mergefailmsg="\
***\n\
*** nightly was unable to automatically merge your changes.  You should\n\
*** redo the full merge manually, following the steps outlined by mercurial\n\
*** above, then restart nightly.\n\
***\n"
        typeset mergepassmsg="\
***\n\
*** nightly successfully merged your changes.  This means that your working\n\
*** directory has been updated, but those changes are not yet committed.\n\
*** After nightly completes, you should validate the results of the merge,\n\
*** then use hg commit manually.\n\
***\n"
```

```
        #
        # For each repository in turn:
        #
        # 1. Do the pull.  If this fails, dump the output and bail out.
        #
        # 2. If the pull resulted in an extra head, do an explicit merge.
        #    If this fails, dump the output and bail out.
        #
        # Because we can't rely on Mercurial to exit with a failure code
        # when a merge fails (Mercurial issue #186), we must grep the
        # output of pull/merge to check for attempted and/or failed merges.
        #
        # 3. If a merge failed, set the message and fail the bringover.
        #
        # 4. Otherwise, if a merge succeeded, set the message
        #
        # 5. Dump the output, and any message from step 3 or 4.
        #

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

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

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

        if grep "not updating" $TMPDIR/pull_open.out > /dev/null 2>&1; then
                staffer hg --cwd $CODEMGR_WS merge \
                    >> $TMPDIR/pull_open.out 2>&1
                if (( $? != 0 )); then
                        printf "%s: merge failed as follows:\n\n" \
                            "$CODEMGR_WS"
                        cat $TMPDIR/pull_open.out
                        if grep "^merging.*failed" $TMPDIR/pull_open.out \
                            > /dev/null 2>&1; then
                                printf "$mergefailmsg"
                        fi
                        touch $TMPDIR/bringover_failed
                        return
                fi
        fi
```

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

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

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

                        #
                        # If there are no incoming changesets, then incoming wil
                        # fail, and there will be no bundle file.  Reset the sou
                        # to allow the remaining logic to complete with no false
                        # negatives.  (Unlike incoming, pull will return success
                        # for the no-change case.)
                        #
                        if (( $? != 0 )); then
                                HG_SOURCE=$CLOSED_BRINGOVER_WS
                        fi
                fi

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

                if grep "not updating" $TMPDIR/pull_closed.out > /dev/null 2>&1;
                        staffer hg --cwd $CODEMGR_WS/usr/closed merge \
                            >> $TMPDIR/pull_closed.out 2>&1
                        if (( $? != 0 )); then
                                printf "closed merge failed as follows:\n\n"
                                cat $TMPDIR/pull_closed.out
                                if grep "^merging.*failed" $TMPDIR/pull_closed.o
                                        printf "$mergefailmsg"
                                fi
                                touch $TMPDIR/bringover_failed
                                return
                        fi
                fi

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

```
                        printf "$mergepassmsg"
                fi
        fi

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

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

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

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

______***unchanged_portion_omitted_***

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

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

        run_hook PRE_BRINGOVER

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

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

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

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

        run_hook POST_BRINGOVER
```

```
        #
        # Possible transition from pre-split workspace to split
        # workspace.  See if the bringover changed anything.
        #
        CLOSED_IS_PRESENT="$orig_closed_is_present"
        check_closed_tree

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

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

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

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

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

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

#
# Report the compiler versions.
#

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

( cd $SRC
  for target in cc-version cc64-version java-version; do
        echo
        #
        # Put statefile somewhere we know we can write to rather than trip
        # over a read-only $srcroot.
        #
        rm -f $TMPDIR/make-state
        export SRC
        if $MAKE -K $TMPDIR/make-state -e $target 2>/dev/null; then
                continue
        fi
        touch $TMPDIR/nocompiler
  done
  echo
) | tee -a $build_environ_file >> $LOGFILE
```

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

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

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

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

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

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

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

        normal_build
fi

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

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

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

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

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


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

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

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

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

fi

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

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

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

        normal_build
fi

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

if [ "$build_ok" = "y" ]; then
        echo "\n==== Creating protolist system file at ‘date‘ ====" \
                >> $LOGFILE
        protolist $checkroot > $ATLOG/proto_list_${MACH}
        echo "==== protolist system file created at ‘date‘ ====\n" \
                >> $LOGFILE

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

                E1=
                f1=
                if [ -d "$SRC/pkgdefs" ]; then
                        f1="$SRC/pkgdefs/etc/exception_list_$MACH"
                        if [ "$X_FLAG" = "y" ]; then
                                f1="$f1 $IA32_IHV_WS/usr/src/pkgdefs/etc/excepti
                        fi
                fi

                for f in $f1; do
                        if [ -f "$f" ]; then
                                E1="$E1 -e $f"
                        fi
                done

                E2=
                f2=
                if [ -d "$SRC/pkg" ]; then
                        f2="$f2 exceptions/packaging"
                fi

                for f in $f2; do
                        if [ -f "$f" ]; then
                                E2="$E2 -e $f"
                        fi
                done
```

```
                if [ -f "$REF_PROTO_LIST" ]; then
                        #
                        # For builds that copy the IHV proto area (-X), add the
                        # IHV proto list to the reference list if the reference
                        # was built without -X.
                        #
                        # For builds that don’t copy the IHV proto area, add the
                        # IHV proto list to the build’s proto list if the
                        # reference was built with -X.
                        #
                        # Use the presence of the first file entry of the cached
                        # IHV proto list in the reference list to determine
                        # whether it was built with -X or not.
                        #
                        IHV_REF_PROTO_LIST=$SRC/pkg/proto_list_ihv_$MACH
                        grepfor=$(nawk ’$1 == "f" { print $2; exit }’ \
                                $IHV_REF_PROTO_LIST 2> /dev/null)
                        if [ $? = 0 -a -n "$grepfor" ]; then
                                if [ "$X_FLAG" = "y" ]; then
                                        grep -w "$grepfor" \
                                                $REF_PROTO_LIST > /dev/null
                                        if [ ! "$?" = "0" ]; then
                                                REF_IHV_PROTO="-d $IHV_REF_PROTO
                                        fi
                                else
                                        grep -w "$grepfor" \
                                                $REF_PROTO_LIST > /dev/null
                                        if [ "$?" = "0" ]; then
                                                IHV_PROTO_LIST="$IHV_REF_PROTO_L
                                        fi
                                fi
                        fi
                fi
        fi

        if [ "$N_FLAG" != "y" -a -f $SRC/pkgdefs/Makefile ]; then
                echo "\n==== Impact on SVr4 packages ====\n" >> $mail_msg_file
                #
                # Compare the build’s proto list with current package
                # definitions to audit the quality of package
                # definitions and makefile install targets. Use the
                # current exception list.
                #
                PKGDEFS_LIST=""
                for d in $abssrcdirs; do
                        if [ -d $d/pkgdefs ]; then
                                PKGDEFS_LIST="$PKGDEFS_LIST -d $d/pkgdefs"
                        fi
                done
                if [ "$X_FLAG" = "y" -a \
                    -d $IA32_IHV_WS/usr/src/pkgdefs ]; then
                        PKGDEFS_LIST="$PKGDEFS_LIST -d $IA32_IHV_WS/usr/src/pkgd
                fi
                $PROTOCMPTERSE \
                    "Files missing from the proto area:" \
                    "Files missing from packages:" \
                    "Inconsistencies between pkgdefs and proto area:" \
                    ${E1} \
                    ${PKGDEFS_LIST} \
                    $ATLOG/proto_list_${MACH} \
                    >> $mail_msg_file
        fi

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

        fi

        if [ "$N_FLAG" != "y" -a -f "$REF_PROTO_LIST" ]; then
                echo "\n==== Impact on proto area ====\n" >> $mail_msg_file
                if [ -n "$E2" ]; then
                        ELIST=$E2
                else
                        ELIST=$E1
                fi
                $PROTOCMPTERSE \
                        "Files in yesterday's proto area, but not today's:" \
                        "Files in today's proto area, but not yesterday's:" \
                        "Files that changed between yesterday and today:" \
                        ${ELIST} \
                        -d $REF_PROTO_LIST \
                        $REF_IHV_PROTO \
                        $ATLOG/proto_list_${MACH} \
                        $IHV_PROTO_LIST \
                        >> $mail_msg_file
        fi
fi

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

# Update parent proto area if necessary. This is done now
# so that the proto area has either DEBUG or non-DEBUG kernels.
# Note that this clears out the lock file, so we can dispense with
# the variable now.
if [ "$U_FLAG" = "y" -a "$build_ok" = "y" ]; then
        echo "\n==== Copying proto area to $NIGHTLY_PARENT_ROOT ====\n" | \
            tee -a $LOGFILE >> $mail_msg_file
        rm -rf $NIGHTLY_PARENT_ROOT/*
        unset Ulockfile
        mkdir -p $NIGHTLY_PARENT_ROOT
        if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
                ( cd $ROOT; tar cf - . |
                    ( cd $NIGHTLY_PARENT_ROOT;  umask 0; tar xpf - ) ) 2>&1 |
                    tee -a $mail_msg_file >> $LOGFILE
        fi
        if [[ "$MULTI_PROTO" = yes && "$F_FLAG" = n ]]; then
                rm -rf $NIGHTLY_PARENT_ROOT-nd/*
                mkdir -p $NIGHTLY_PARENT_ROOT-nd
                cd $ROOT-nd
                ( tar cf - . |
                    ( cd $NIGHTLY_PARENT_ROOT-nd; umask 0; tar xpf - ) ) 2>&1 |
                    tee -a $mail_msg_file >> $LOGFILE
        fi
        if [ -n "${NIGHTLY_PARENT_TOOLS_ROOT}" ]; then
                echo "\n==== Copying tools proto area to $NIGHTLY_PARENT_TOOLS_R
                    tee -a $LOGFILE >> $mail_msg_file
                rm -rf $NIGHTLY_PARENT_TOOLS_ROOT/*
                mkdir -p $NIGHTLY_PARENT_TOOLS_ROOT
                if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
                        ( cd $TOOLS_PROTO; tar cf - . |
                            ( cd $NIGHTLY_PARENT_TOOLS_ROOT;
                            umask 0; tar xpf - ) ) 2>&1 |
                            tee -a $mail_msg_file >> $LOGFILE
                fi
        fi
fi
```

```

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

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

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

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

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

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

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

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

                fi
        fi

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

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

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

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

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

		if [[ -f $elf_ddir.ref/runtime.attr ]]; then
			diff $elf_ddir.ref/runtime.attr \
				$elf_ddir/runtime.attr \
				>> $mail_msg_file
		fi
	fi

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

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

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

# DEBUG lint of kernel begins
```

```
if [ "$i_CMD_LINE_FLAG" = "n" -a "$l_FLAG" = "y" ]; then
	if [ "$LINTDIRS" = "" ]; then
		# LINTDIRS="$SRC/uts y $SRC/stand y $SRC/psm y"
		LINTDIRS="$SRC y"
	fi
	set $LINTDIRS
	while [ $# -gt 0 ]; do
		dolint $1 $2; shift; shift
	done
else
	echo "\n==== No '$MAKE lint' ====\n" >> $LOGFILE
fi

# "make check" begins

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

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

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

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

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

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

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

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

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

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

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

```
if [ "$SO_FLAG" = n -a "$O_FLAG" = y -a "$build_ok" = y ]; then
        echo "\n==== Generating OpenSolaris tarballs ====\n" | \
            tee -a $mail_msg_file >> $LOGFILE

        cd $CODEMGR_WS

        #
        # This step grovels through the package manifests, so it
        # must come after findunref.
        #
        # We assume no DEBUG vs non-DEBUG package content variation
        # here; if that changes, then the "make all" in $SRC/pkg will
        # need to be moved into the conditionals and repeated for each
        # different build.
        #
        echo "Generating closed binaries tarball(s)..." >> $LOGFILE
        closed_basename=on-closed-bins
        if [ "$D_FLAG" = y ]; then
                bindrop "$closed_basename" >>"$LOGFILE" 2>&1
                if (( $? != 0 )) ; then
                        echo "Couldn't create DEBUG closed binaries." |
                            tee -a $mail_msg_file >> $LOGFILE
                        build_ok=n
                fi
        fi
        if [ "$F_FLAG" = n ]; then
                bindrop -n "$closed_basename-nd" >>"$LOGFILE" 2>&1
                if (( $? != 0 )) ; then
                        echo "Couldn't create non-DEBUG closed binaries." |
                            tee -a $mail_msg_file >> $LOGFILE
                        build_ok=n
                fi
        fi

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

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

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

        abspkgdefs=
        abspkg=
        for d in $abssrcdirs; do
                if [ -d "$d/pkgdefs" ]; then
                        abspkgdefs="$abspkgdefs $d"
                fi
```

```
                if [ -d "$d/pkg" ]; then
                        abspkg="$abspkg $d"
                fi
        done

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

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

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

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

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

typeset -i10 hours
typeset -Z2 minutes
typeset -Z2 seconds

elapsed_time=$SECONDS
((hours = elapsed_time / 3600 ))
((minutes = elapsed_time / 60  % 60))
((seconds = elapsed_time % 60))

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

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

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

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

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

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

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

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

if [[ "$1" = "-e" ]]; then
        setenv=true
        shift
else
        setenv=false
fi

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

#
# No workspace/repository path was given, so try and detect one from our
# current directory we’re in
#
if [[ $# -lt 1 ]]; then
        if env CODEMGR_WS="" $WHICH_SCM | read SCM_MODE tmpwsname && \
            [[ $SCM_MODE != unknown ]]; then
                echo "Defaulting to $SCM_MODE repository $tmpwsname"
        else
                echo "usage: ws [-e] [workspace_name]" >&2
                if $setenv; then
                        cleanup_env
                        return 1
                else
                        exit 1
                fi
        fi
else
        #
        # A workspace/repository path was passed in, grab it and pop
        # it off the stack
        #
        tmpwsname=$1
        shift
fi

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

#
# Set the workspace name and unset tmpwsname (as we reuse it later)
#
wsname=‘echo $tmpwsname|fmtwsname‘
unset tmpwsname

#
# Checking for CODEMGR_WSPATH
#
if [[ -n ${CODEMGR_WSPATH} && ( ! -d $wsname ) && \
     ( ‘expr "$wsname" : "\/"‘ = "0" ) ]]
then
        ofs=$IFS
        IFS=":  "
        for i in $CODEMGR_WSPATH
```

```
        do
                if [[ -d ${i}/${wsname} ]]; then
                        wsname=${i}/${wsname}
                        break
                fi
        done
        IFS=$ofs
fi

#
# to translate it to an absolute pathname.  We need an
# absolute pathname in order to set CODEMGR_WS.
#
if [[ ‘expr "$wsname" : "\/"‘ = "0" ]]
then
        pwd=‘pwd‘
        wsname="$pwd/$wsname"
fi

#
#       Check to see if this is a valid workspace
#
if [[ ! -d $wsname ]]; then
        echo "$wsname . . . no such directory" >&2
        if $setenv; then
                cleanup_env
                return 1
        else
                exit 1
        fi
fi

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

wsname=$tmpwsname
CODEMGR_WS=$wsname ; export CODEMGR_WS
SRC=$wsname/usr/src; export SRC
TSRC=$wsname/usr/ontest; export TSRC

if [[ "$SCM_MODE" = "teamware" && -d ${wsname}/Codemgr_wsdata ]]; then
        CM_DATA="Codemgr_wsdata"
        wsosdir=$CODEMGR_WS/$CM_DATA/sunos
        protofile=$wsosdir/protodefs
elif [[ "$SCM_MODE" = "mercurial" && -d ${wsname}/.hg ]]; then
        CM_DATA=".hg"
        wsosdir=$CODEMGR_WS/$CM_DATA
        protofile=$wsosdir/org.opensolaris.protodefs
elif [[ "$SCM_MODE" = "git" && -d ${wsname}/.git ]]; then
        CM_DATA=".git"
        wsosdir=$CODEMGR_WS/$CM_DATA
        protofile=$wsosdir/org.opensolaris.protodefs
else
        echo "$wsname is not a supported workspace; type is $SCM_MODE" >&2
        if $setenv; then
                cleanup_env
                return 1
        else
                exit 1
        fi
fi

MACH=‘uname -p‘

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

                # Just make one in /tmp
                wsosdir=/tmp
                protofile=$wsosdir/protodefs
        fi

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

        cat << PROTOFILE_EoF > $protofile
#!/bin/sh
#
#       Set default proto areas for this workspace
#       NOTE: This file was initially automatically generated.
#
#       Feel free to edit this file.  If this file is removed
#       it will be rebuilt containing default values.
#
#       The variable CODEMGR_WS is available to this script.
#
#       PROTO1 is the first proto area searched and is typically set
#       to a proto area associated with the workspace.  The ROOT
#       environment variable is set to the same as PROTO1.  If you
#       will be doing make installs this proto area needs to be writable.
#
#       PROTO2 and PROTO3 are set to proto areas to search before the
#       search proceeds to the local machine or the proto area specified by
#       TERMPROTO.
#
#       TERMPROTO (if specified) is the last place searched.  If
#       TERMPROTO is not specified the search will end at the local
#       machine.
#

PROTO1=\$CODEMGR_WS/proto
PROTOFILE_EoF

        if [[ "$SCM_MODE" = "teamware" ]]; then
                cat << PROTOFILE_EoF >> $protofile
if [[ -f "\$CODEMGR_WS/Codemgr_wsdata/parent" ]]; then
   #
   # If this workspace has an codemgr parent then set PROTO2 to
   # point to the parents proto space.
   #
   parent=\‘workspace parent \$CODEMGR_WS\‘
   if [[ -n \$parent ]]; then
           PROTO2=\$parent/proto
   fi
```

```
fi
PROTOFILE_EoF
        elif [[ "$SCM_MODE" = "mercurial" ]]; then
                cat << PROTOFILE_EoF >> $protofile
parent=\‘(cd \$CODEMGR_WS && hg path default 2>/dev/null)\‘
if [[ \$? -eq 0 && -n \$parent ]]; then
   [[ -n \$(check_proto \$parent/proto) ]] && PROTO2=\$parent/proto
fi
PROTOFILE_EoF
        fi
fi

. $protofile

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

MAKEFLAGS=e; export MAKEFLAGS

#
#       Set up the environment variables
#
ROOT=/proto/root_${MACH}        # default

ENVCPPFLAGS1=
ENVCPPFLAGS2=
ENVCPPFLAGS3=
ENVCPPFLAGS4=
ENVLDLIBS1=
ENVLDLIBS2=
ENVLDLIBS3=

PROTO1=‘check_proto $PROTO1‘
if [[ -n "$PROTO1" ]]; then     # first proto area specifed
        ROOT=$PROTO1
        ENVCPPFLAGS1=-I$ROOT/usr/include
        export ENVCPPFLAGS1
        ENVLDLIBS1="-L$ROOT/lib -L$ROOT/usr/lib"
        export ENVLDLIBS1

        PROTO2=‘check_proto $PROTO2‘
        if [[ -n "$PROTO2" ]]; then     # second proto area specifed
                ENVCPPFLAGS2=-I$PROTO2/usr/include
                export ENVCPPFLAGS2
                ENVLDLIBS2="-L$PROTO2/lib -L$PROTO2/usr/lib"
                export ENVLDLIBS2

                PROTO3=‘check_proto $PROTO3‘
                if [[ -n "$PROTO3" ]]; then     # third proto area specifed
                        ENVCPPFLAGS3=-I$PROTO3/usr/include
                        export ENVCPPFLAGS3
                        ENVLDLIBS3="-L$PROTO3/lib -L$PROTO3/usr/lib"
                        export ENVLDLIBS3
                fi
        fi
fi

export ROOT

if [[ -n "$TERMPROTO" ]]; then  # fallback area specifed
        TERMPROTO=‘check_proto $TERMPROTO‘
        ENVCPPFLAGS4="-Y I,$TERMPROTO/usr/include"
        export ENVCPPFLAGS4
        ENVLDLIBS3="$ENVLDLIBS3 -Y P,$TERMPROTO/lib:$TERMPROTO/usr/lib"
        export ENVLDLIBS3
fi
```

```
osbld_flag=0

if [[ ! -v CLOSED_IS_PRESENT ]]; then
        if [[ -d $SRC/../closed ]]; then
                export CLOSED_IS_PRESENT="yes"
        else
                export CLOSED_IS_PRESENT="no"
        fi
fi

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

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

export PATH

if [[ -n "$PROTO2" ]]; then
   # This should point to the parent's proto
   PARENT_ROOT=$PROTO2
   export PARENT_ROOT
else
   # Clear it in case it's already in the env.
   PARENT_ROOT=
fi
export ONBLD_DIR
export MACH

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

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

echo ""
echo "Workspace                    : $wsname"
if [[ -n "$parent" ]]; then
   echo "Workspace Parent             : $parent"
fi
echo "Proto area (\$ROOT)           : $ROOT"
if [[ -n "$PARENT_ROOT" ]]; then
   echo "Parent proto area (\$PARENT_ROOT) : $PARENT_ROOT"
fi
echo "Root of source (\$SRC)        : $SRC"
echo "Root of test source (\$TSRC)  : $TSRC"
if [[ $osbld_flag = "1" ]]; then
   echo "Prepended to PATH            : $ONBLD_DIR"
fi
echo "Current directory (\$PWD)     : $wsname"
echo ""

cd $wsname

if $setenv; then
        cleanup_env
else
        exec ${SHELL:-sh} "$@"
```

```
fi
```