```
**********************************************************
   10689 Sun Jan 26 21:59:57 2014
new/usr/src/pkg/manifests/developer-build-onbld.mf
4525 remove last vestiges of tonic
**********************************************************
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
  24 # Copyright 2010, Richard Lowe
  25 # Copyright 2012, Piotr Jasiukajtis
  26 #

  28 set name=pkg.fmri value=pkg:/developer/build/onbld@$(PKGVERS)
  29 set name=pkg.description value="tools used to build the OS-Net consolidation"
  30 set name=pkg.summary value="OS-Net Build Tools"
  31 set name=info.classification \
  32     value="org.opensolaris.category.2008:Development/Distribution Tools"

  34 #
  35 # This package should not be incorporated.  This allows the tools
  36 # to be upgraded without upgrading the entire system.
  37 #
  38 set name=org.opensolaris.noincorp value=true
  39 set name=variant.arch value=$(ARCH)
  40 dir path=opt group=sys
  41 dir path=opt/onbld
  42 dir path=opt/onbld/bin
  43 dir path=opt/onbld/bin/$(ARCH)
  44 dir path=opt/onbld/env
  45 dir path=opt/onbld/etc
  46 dir path=opt/onbld/etc/exception_lists
  47 dir path=opt/onbld/gk
  48 dir path=opt/onbld/lib
  49 dir path=opt/onbld/lib/$(ARCH)
  50 dir path=opt/onbld/lib/perl
  51 dir path=opt/onbld/lib/python2.6
  52 dir path=opt/onbld/lib/python2.6/onbld
  53 dir path=opt/onbld/lib/python2.6/onbld/Checks
  54 dir path=opt/onbld/lib/python2.6/onbld/Scm
  55 dir path=opt/onbld/lib/python2.6/onbld/hgext
  56 dir path=opt/onbld/man
  57 dir path=opt/onbld/man/man1
  58 $(i386_ONLY)file path=opt/onbld/bin/$(ARCH)/aw mode=0555
  59 $(sparc_ONLY)file path=opt/onbld/bin/$(ARCH)/chk4ubin mode=0555
  60 file path=opt/onbld/bin/$(ARCH)/codereview mode=0555
  61 file path=opt/onbld/bin/$(ARCH)/cscope-fast mode=0555
```

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

```
file path=opt/onbld/bin/wsdiff mode=0555
file path=opt/onbld/bin/xref mode=0555
file path=opt/onbld/bin/xref.mk
file path=opt/onbld/env/developer
file path=opt/onbld/env/gatekeeper
file path=opt/onbld/env/illumos
file path=opt/onbld/etc/SampleLinks
file path=opt/onbld/etc/SamplePkgLinks
file path=opt/onbld/etc/exception_lists/check_rtime
file path=opt/onbld/etc/exception_lists/interface_check
file path=opt/onbld/etc/exception_lists/interface_cmp
file path=opt/onbld/etc/hgstyle
file path=opt/onbld/etc/its.conf
file path=opt/onbld/etc/its.reg
file path=opt/onbld/gk/.cshrc
file path=opt/onbld/gk/.login
file path=opt/onbld/gk/gen_make.machines mode=0755
file path=opt/onbld/lib/$(ARCH)/libdwarf.so.1
file path=opt/onbld/lib/perl/onbld_elfmod.pm
file path=opt/onbld/lib/perl/onbld_elfmod_vertype.pm
file path=opt/onbld/lib/python2.6/onbld/Checks/CStyle.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/CStyle.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/Cddl.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/Cddl.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/CmtBlk.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/CmtBlk.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/Comments.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/Comments.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/Copyright.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/Copyright.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/DbLookups.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/DbLookups.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/HdrChk.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/HdrChk.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/JStyle.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/JStyle.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/Keywords.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/Keywords.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/Mapfile.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/Mapfile.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/ProcessCheck.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/ProcessCheck.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/__init__.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Checks/__init__.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Scm/Backup.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Scm/Backup.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Scm/Version.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Scm/Version.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Scm/WorkSpace.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Scm/WorkSpace.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/Scm/__init__.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/Scm/__init__.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/__init__.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/__init__.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/hgext/__init__.py mode=0444
file path=opt/onbld/lib/python2.6/onbld/hgext/__init__.pyc mode=0444
file path=opt/onbld/lib/python2.6/onbld/hgext/cdm.py mode=0444
file path=opt/onbld/man/man1/Install.1
file path=opt/onbld/man/man1/bldenv.1
file path=opt/onbld/man/man1/bringovercheck.1
file path=opt/onbld/man/man1/cddlchk.1
file path=opt/onbld/man/man1/check_rtime.1
file path=opt/onbld/man/man1/checkpaths.1
file path=opt/onbld/man/man1/codereview.1
file path=opt/onbld/man/man1/cstyle.1
file path=opt/onbld/man/man1/cw.1
```

```
file path=opt/onbld/man/man1/find_elf.1
file path=opt/onbld/man/man1/findunref.1
file path=opt/onbld/man/man1/flg.flp.1
file path=opt/onbld/man/man1/get_depend_info.1
file path=opt/onbld/man/man1/git-pbchk.1
file path=opt/onbld/man/man1/hdrchk.1
file path=opt/onbld/man/man1/hgsetup.1
file path=opt/onbld/man/man1/interface_check.1
file path=opt/onbld/man/man1/interface_cmp.1
file path=opt/onbld/man/man1/jstyle.1
file path=opt/onbld/man/man1/lintdump.1
file path=opt/onbld/man/man1/make_pkg_db.1
file path=opt/onbld/man/man1/mapfilechk.1
file path=opt/onbld/man/man1/ndrgen.1
file path=opt/onbld/man/man1/nightly.1
file path=opt/onbld/man/man1/onu.1
file path=opt/onbld/man/man1/sccscheck.1
file path=opt/onbld/man/man1/signit.1
file path=opt/onbld/man/man1/signproto.1
file path=opt/onbld/man/man1/webrev.1
file path=opt/onbld/man/man1/which_scm.1
file path=opt/onbld/man/man1/ws.1
file path=opt/onbld/man/man1/wsdiff.1
file path=opt/onbld/man/man1/xref.1
hardlink path=opt/onbld/bin/$(ARCH)/install.bin target=./install
legacy pkg=SUNWonbld desc="tools used to build the OS-Net consolidation" \
    name="OS-Net Build Tools" version=11.11,REV=2009.10.22
license cr_Sun license=cr_Sun
license lic_CDDL license=lic_CDDL
license usr/src/tools/ctf/dwarf/THIRDPARTYLICENSE \
    license=usr/src/tools/ctf/dwarf/THIRDPARTYLICENSE
license usr/src/tools/onbld/THIRDPARTYLICENSE \
    license=usr/src/tools/onbld/THIRDPARTYLICENSE
link path=opt/onbld/bin/git-nits target=git-pbchk
link path=opt/onbld/lib/python target=python2.6
link path=opt/onbld/man/man1/git-nits.1 target=git-pbchk.1
# webrev(1) requires ps2pdf
depend fmri=print/filter/ghostscript type=require
# hgsetup(1) uses check-hostname(1) and nightly sendmail(1M)
depend fmri=service/network/smtp/sendmail type=require
# nightly(1) uses wget
depend fmri=web/wget type=require
```

****
1174 Sun Jan 26 21:59:58 2014
new/usr/src/req.flg
4525 remove last vestiges of tonic
****

```
#!/bin/sh
#
# 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.
#

echo_file usr/src/Makefile
echo_file usr/src/Targetdirs
echo_file usr/src/Makefile.master
echo_file usr/src/Makefile.noget
echo_file usr/src/Makefile.master.64
echo_file usr/src/Makefile.msg.targ
echo_file usr/src/Makefile.psm
echo_file usr/src/Makefile.psm.targ
echo_file usr/closed/cmd/cmd-crypto/etc/certs/SUNWosnetCF
echo_file usr/closed/cmd/cmd-crypto/etc/certs/SUNWosnetSE
echo_file usr/closed/cmd/cmd-crypto/etc/keys/SUNWosnetCF
echo_file usr/closed/cmd/cmd-crypto/etc/keys/SUNWosnetSE
```

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

```
#
# usage: fixcrypto listfile ctop
# Massage entries in listfile for crypto modules, so that they point
# into ctop.
#
function fixcrypto {
        typeset listfile=$1
        typeset ctop=$2

        typeset ccontents=/tmp/crypto-toc$$
        find "$ctop" -type f -print > $ccontents
        typeset root=root_$MACH
        [ "$OBJD" = obj ] && root=root_$MACH-nd

        grep -v ^MOD $listfile > $listfile.no-mod
        grep ^MOD $listfile | while read tag srcdir module targdir size impl; do
                #
                # We don't just grep for ${OBJD}$size/$module because
                # there can be generic and platform-dependent versions
                # of a module.
                #
                newsrcfile=$(grep -w $root/$targdir/${OBJD}$size/$module $cconte
                if [ -n "$newsrcfile" ]; then
                        # srcdir doesn't include final objNN or debugNN
                        echo $tag $module $targdir $size $impl \
                            $(dirname $(dirname "$newsrcfile"))
                else
                        echo $tag $module $targdir $size $impl $srcdir
                fi
        done > $listfile.mod
        cat $listfile.mod $listfile.no-mod > $listfile

        rm -f $listfile.mod
        rm -f $listfile.no-mod
        rm -f $ccontents
}

#
# Copy a module, or create a link, as needed.
#

function copymod {
        case $1 in
        MOD)
                targdir=$INSTALL_FILES/$4
                tstmkdir $targdir
                target=$targdir/$3
                verbose "$INSTALL_CP $2/${OBJD}$5/$3 $target"
                $INSTALL_CP $2/${OBJD}$5/$3 $target || \
                    fail "can't create $target"
                ;;
        SYMLINK)
                targdir=$INSTALL_FILES/$4
                tstmkdir $targdir
                target=$targdir/$5
                rm -f $target
                verbose "ln -s $3 $target"
                ln -s $3 $target || fail "can't create $target"
                ;;
```

```
        LINK)
                targdir=$INSTALL_FILES/$5
                tstmkdir $targdir
                target=$targdir/$6
                rm -f $target
                verbose "ln $INSTALL_FILES/$3/$4 $target"
                ln $INSTALL_FILES/$3/$4 $target || fail "can't create $target"
                ;;
        CONF)
                target=$INSTALL_FILES/$3
                tstmkdir `dirname $target`
                conffile=`basename $3`
                verbose "$INSTALL_CP $4/$conffile $target"
                $INSTALL_CP $4/$conffile $target
                ;;
        *)
                fail "unrecognized modlist entry: $*"
                ;;
        esac
}
```

______unchanged_portion_omitted_

```
#
# Copy kernel modules to $INSTALL_DIR
#

function copy_kernel {

        case $KARCH in
                sun4*)          ISA=sparc;      MACH=sparc      ;;
                i86*)           ISA=intel;      MACH=i386       ;;
                *)              fail "${KARCH}: invalid kernel architecture";;
        esac
        export MACH

        if [ "$GLOM" = "no" ]; then
                verbose "Source = $UTS, ISA = $ISA, kernel = $KARCH"
        else
                verbose "Source = $UTS, ISA = $ISA, kernel = $KARCH, impl = $IMP
        fi

        test -d $KARCH || fail "${KARCH}: invalid kernel architecture"
        test -d $ISA || fail "${ISA}: invalid instruction set architecture"

        tstmkdir $INSTALL_FILES
        rm -rf $modstatedir
        tstmkdir $modstatedir
        export MODSTATE=$modstatedir/state

        #
        # Figure out which "make" to use.  dmake is faster than serial
        # make, but dmake 7.3 has a bug that causes it to lose log
        # output, which means the modlist might be incomplete.
        #
        make=dmake
        dmvers=`$make -version`
        if [ $? -ne 0 ]; then
                make=/usr/ccs/bin/make
        elif [[ $dmvers = *Distributed?Make?7.3* ]]; then
                unset make
                searchpath="/ws/onnv-tools/SUNWspro/SOS10/bin
                        /opt/SUNWspro/SOS10/bin
                        /opt/SUNWspro/bin"
                for dmpath in $searchpath; do
                        verbose "Trying $dmpath/dmake"
                        if [ -x $dmpath/dmake ]; then
```

```
				dmvers=‘$dmpath/dmake -version‘
				if [[ $dmvers != *Distributed?Make?7.3* ]]; then
					make="$dmpath/dmake"
					break;
				fi
			fi
		done
		if [ -z $make ]; then
			make=/usr/ccs/bin/make
			echo "Warning: dmake 7.3 doesn’t work with Install;" \
				"using $make"
		fi
	fi

	#
	# Get a list of all modules, configuration files, and links
	# that we might want to install.
	#
	verbose "Building module list..."
	(cd $KARCH; MAKEFLAGS=e $make -K $MODSTATE modlist.karch) | \
	    egrep "^MOD|^CONF|^LINK|^SYMLINK" > $modlist
	[ "$VERBOSE" = "V" ] && cat $modlist
	check_modlist $modlist
	if [ -n "$ON_CRYPTO_BINS" ]; then
		cryptotar="$ON_CRYPTO_BINS"
		if [ "$OBJD" = obj ]; then
			isa=$(uname -p)
			cryptotar=$(echo "$ON_CRYPTO_BINS" |
			    sed -e s/.$isa.tar.bz2/-nd.$isa.tar.bz2/)
		fi
		[ -f "$cryptotar" ] || fail "crypto ($cryptotar) doesn’t exist"
		cryptotree=$(mktemp -d /tmp/crypto.XXXXXX)
		[ -n "$cryptotree" ] || fail "can’t create tree for crypto"
		unpack_crypto "$cryptotar" "$cryptotree"
		#
		# fixcrypto must come before fixglom, because
		# fixcrypto uses the unglommed path to find things in
		# the unpacked crypto.
		#
		fixcrypto $modlist "$cryptotree"
	fi
	if [ "$GLOM" = "yes" ]; then
		fixglom $modlist $GLOMNAME
		filtimpl $modlist $IMPL
	fi
	if [[ -n "$files" && "$files" != All ]]; then
		filtmod $modlist "$files"
	fi

	#
	# Copy modules and create links.  For architectures with both
	# 32- and 64-bit modules, we’ll likely have duplicate
	# configuration files, so do those after filtering out the
	# duplicates.
	#
	verbose "Copying files to ${INSTALL_FILES}..."

	#
	# The IFS is reset to the newline character so we can buffer the
	# output of grep without piping it directly to copymod, otherwise
	# if fail() is called, then it will deadlock in fail()’s wait call
	#
	OIFS="$IFS"
	IFS="
	"
	set -- ‘grep -v "^CONF" $modlist‘;
```

```
	IFS="$OIFS"
	for onemod in "$@"; do
		copymod $onemod
	done

	OIFS="$IFS"
	IFS="
	"
	set -- ‘grep "^CONF" $modlist | sort | uniq‘;
	IFS="$OIFS"
	for onemod in "$@"; do
		copymod $onemod
	done

	#
	# Add the glommed kernel name to the root archive
	#
	if [[ $GLOM == "yes" ]];
	then
		filelist="$INSTALL_FILES/etc/boot/solaris/filelist.ramdisk"
		mkdir -p ‘dirname $filelist‘
		echo "platform/$KARCH/$GLOMNAME" >$filelist
	fi

	STATE=1 # all kernel modules copied correctly
	save_state
}
```

______***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=$SRC
        if [ -z "$ON_CRYPTO_BINS" ]; then
                echo "Warning: ON_CRYPTO_BINS not set; pre-signed" \
                    "crypto not provided."
        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_***

```
**********************************************************
    3584 Sun Jan 26 21:59:59 2014
new/usr/src/tools/scripts/Makefile
4525 remove last vestiges of tonic
**********************************************************
#
# 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.
#
# Copyright 2010, Richard Lowe

SHELL=/usr/bin/ksh93

SHFILES= \
        Install \
        bindrop \
        bldenv \
        build_cscope \
        bringovercheck \
        checkpaths \
        checkproto \
        cryptodrop \
        cstyle \
        elfcmp \
        flg.flp \
        genoffsets \
        hgsetup \
        nightly \
        onu \
        protocmp.terse \
        sccscheck \
        webrev \
        which_scm \
        ws \
        xref

PERLFILES= \
        check_rtime \
        find_elf \
        interface_check \
        interface_cmp \
        jstyle \
        mkreadme_osol \
        mktpl \
        validate_flg \
        validate_paths \
        wdiff
```

```
PERLMODULES= \
        onbld_elfmod.pm \
        onbld_elfmod_vertype.pm


PYFILES= \
        cddlchk \
        copyrightchk \
        git-pbchk \
        hdrchk \
        hg-active \
        mapfilechk \
        validate_pkg \
        wsdiff

SCRIPTLINKS= \
        git-nits

MAN1FILES= \
        Install.1 \
        bldenv.1 \
        bringovercheck.1 \
        cddlchk.1 \
        checkpaths.1 \
        check_rtime.1 \
        cstyle.1 \
        find_elf.1 \
        flg.flp.1 \
        git-pbchk.1 \
        hdrchk.1 \
        interface_check.1 \
        interface_cmp.1 \
        hgsetup.1 \
        jstyle.1 \
        mapfilechk.1 \
        nightly.1 \
        onu.1 \
        sccscheck.1 \
        webrev.1 \
        which_scm.1 \
        ws.1 \
        wsdiff.1 \
        xref.1

MAN1LINKS= \
        git-nits.1

MAKEFILES= \
        xref.mk

ETCFILES= \
        hgstyle \
        its.conf \
        its.reg

EXCEPTFILES= \
        check_rtime \
        interface_check \
        interface_cmp

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

include ../Makefile.tools

ROOTONBLDSCRIPTLINKS = $(SCRIPTLINKS:%=$(ROOTONBLDBIN)/%)
```

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

$(ROOTONBLDETCFILES)    := FILEMODE=    644
$(ROOTONBLDEXCEPTFILES) := FILEMODE=    644
$(ROOTONBLDPERLMODULES) := FILEMODE=    644
$(ROOTONBLDMAKEFILES)   := FILEMODE=    644
$(ROOTONBLDMAN1FILES)   := FILEMODE=    644

.KEEP_STATE:

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

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

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

install: all .WAIT $(ROOTONBLDSHFILES) $(ROOTONBLDPERLFILES)    \
                $(ROOTONBLDPERLMODULES) $(ROOTONBLDPYFILES)     \
                $(ROOTONBLDSCRIPTLINKS) $(ROOTONBLDMAN1FILES)   \
                $(ROOTONBLDMAKEFILES) $(ROOTONBLDETCFILES)      \
                $(ROOTONBLDEXCEPTFILES) $(ROOTONBLDMAN1LINKS)

clean:
        $(RM) $(CLEANFILES)

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

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

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

include ../Makefile.targ
```

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

```
[+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 ;;
          \?)   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 \
        CH \
#endif /* ! codereview */
```

```
        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="-${NIGHTLY_OPTIONS#-}"
while getopts '+0ABCDdFfGIilMmNnpRrtUuw' FLAG "$NIGHTLY_OPTIONS"
while getopts '+0AaBCDdFfGIilMmNnopRrtUuWwXxz' FLAG "$NIGHTLY_OPTIONS"
do
```

```
        case "$FLAG" in
          o)    flags.o=true  ;;
          +o)   flags.o=false ;;
          t)    flags.t=true  ;;
          +t)   flags.t=false ;;
          *)    ;;
        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

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

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

#       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 [[ -n "${MAKE}" ]]; then
        if [[ -x "${MAKE}" ]]; then
                export PATH="$(dirname -- "${MAKE}"):$PATH"
        else
                print "\$MAKE (${MAKE}) is not a valid executible"
                exit 1
        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}"

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
```

```
**********************************************************
    3633 Sun Jan 26 22:00:00 2014
new/usr/src/tools/scripts/checkpaths.sh
4525 remove last vestiges of tonic
**********************************************************
#!/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
```

```
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 \
                            $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 $SRC/tools/opensolaris/license-list ]; then
        sed -e 's/$/.descrip/' < $SRC/tools/opensolaris/license-list | \
                validate_paths -n SRC/tools/opensolaris/license-list
                validate_paths -n SRC/tools/opensolaris/license-list \
                    -e ^usr/closed
fi

validate_flg -f
# 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

validate_flg $f_flg -e ^usr/closed/

exit 0
```

****
   14823 Sun Jan 26 22:00:01 2014
new/usr/src/tools/scripts/nightly.1
4526 nightly contains a great deal of effectively dead code
****

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

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

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

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

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

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

.LP
.SH ENVIRONMENT VARIABLES
.LP
Here is a list of prominent environment variables that
.I nightly
references and the meaning of each variable.
.LP
.RE
.B CODEMGR_WS
.RS 5
The root of your workspace, including whatever metadata is kept by
the source code management system.  This is the workspace in which the
build will be done.
```

```
.RE
.LP
.B PARENT_WS
.RS 5
The root of the workspace that is the parent of the
one being built.  This is particularly relevant for configurations
with a main
workspace and build workspaces underneath it; see the
\-u and \-U
options as well as the PKGARCHIVE environment variable, for more
information.
.RE
.LP
.B BRINGOVER_WS
.RS 5
This is the workspace from which
.I nightly
will fetch sources to either populate or update your workspace;
it defaults to $CLONE_WS.
.RE
.LP
.B CLOSED_BRINGOVER_WS
.RS 5
A full Mercurial workspace has two repositories: one for open source
and one for closed source.  If this variable is non-null,
.I nightly
will pull from the repository that it names to get the closed source.
It defaults to $CLOSED_CLONE_WS.
.LP
If $CODEMGR_WS already exists and contains only the open repository,
.I nightly
will ignore this variable; you’ll need to pull the closed repository
by hand if you want it.
.RE
.LP
.B CLONE_WS
.RS 5
This is the workspace from which
.I nightly
will fetch sources by default.  This is
often distinct from the parent, particularly if the parent is a gate.
.RE
.LP
.B CLOSED_CLONE_WS
.RS 5
This is the default closed-source Mercurial repository that
.I nightly
might pull from (see
.B CLOSED_BRINGOVER_WS
for details).
.RE
.LP
.B SRC
.RS 5
Root of OS-Net source code, referenced by the Makefiles.  It is
the starting point of build activity.  It should be expressed
in terms of $CODEMGR_WS.
.RE
.LP
.B ROOT
.RS 5
Root of the proto area for the build.  The makefiles direct
installation of build products to this area and
direct references to these files by builds of commands and other
targets.  It should be expressed in terms of $CODEMGR_WS.
.LP
```

```
If $MULTI_PROTO is "no", $ROOT may contain a DEBUG or non-DEBUG
build.  If $MULTI_PROTO is "yes", $ROOT contains the DEBUG build and
$ROOT-nd contains the non-DEBUG build.
.RE
.LP
.B TOOLS_ROOT
.RS 5
Root of the tools proto area for the build.  The makefiles direct
installation of tools build products to this area.  Unless \fB+t\fR
is part of $NIGHTLY_OPTIONS, these tools will be used during the
build.
.LP
As built by nightly, this will always contain non-DEBUG objects.
Therefore, this will always have a -nd suffix, regardless of
$MULTI_PROTO.
.RE
.LP
.B MACH
.RS 5
The instruction set architecture of the build machine as given
by \fIuname -p\fP, e.g. sparc, i386.
.RE
.LP
.B LOCKNAME
.RS 5
The name of the file used to lock out multiple runs of
.IR nightly .
This should generally be left to the default setting.
.RE
.LP
.B ATLOG
.RS 5
The location of the log directory maintained by
.IR nightly .
This should generally be left to the default setting.
.RE
.LP
.B LOGFILE
.RS 5
The name of the log file in the $ATLOG directory maintained by
.IR nightly .
This should generally be left to the default setting.
.RE
.LP
.B STAFFER
.RS 5
The non-root account to use on the build machine for the
bringover from the clone or parent workspace.
This may not be the same identify used by the SCM.
.RE
.LP
.B MAILTO
.RS 5
The address to be used to send completion e-mail at the end of
the build (for the \-m option).
.RE
.LP
.B MAILFROM
.RS 5
The address to be used for From: in the completion e-mail at the
end of the build (for the \-m option).
.RE
.LP
.B REF_PROTO_LIST
.RS 5
Name of file used with protocmp to compare proto area contents.
```

```
.RE
.LP
.B PARENT_ROOT
.RS 5
The parent root, which is the destination for copying the proto
area(s) when using the \-U option.
.RE
.LP
.B PARENT_TOOLS_ROOT
.RS 5
The parent tools root, which is the destination for copying the tools
proto area when using the \-U option.
.RE
.LP
.B RELEASE
.RS 5
The release version number to be used; e.g., 5.10.1 (Note: this is set
in Makefile.master and should not normally be overridden).
.RE
.LP
.B VERSION
.RS 5
The version text string to be used; e.g., "onnv:‘date ’+%Y-%m-%d’‘".
.RE
.LP
.B RELEASE_DATE
.RS 5
The release date text to be used; e.g., October 2009. If not set in
your environment file, then this text defaults to the output from
$(LC_ALL=C date +"%B %Y"); e.g., "October 2009".
.RE
.LP
.B RELEASE_BUILD
.RS 5
Define this to build a release with a non-DEBUG kernel.
Generally, let
.I nightly
set this for you based on its options.
.RE
.LP
.B PKGARCHIVE
.RS 5
The destination for packages.  This may be relative to
$CODEMGR_WS for private packages or relative to $PARENT_WS
if you have different workspaces for different architectures
but want one hierarchy of packages.
.RE
.LP
.B MAKEFLAGS
.RS 5
Set default flags to make; e.g., -k to build all targets regardless of errors.
.RE
.LP
.B UT_NO_USAGE_TRACKING
.RS 5
Disables usage reporting by listed Devpro tools. Otherwise it sends mail
to some Devpro machine every time the tools are used.
.RE
.LP
.B LINTDIRS
.RS 5
Directories to lint with the \-l option.
.RE
.LP
.B BUILD_TOOLS
.RS 5
```

```
BUILD_TOOLS is the root of all tools including the compilers; e.g.,
/ws/onnv-tools.  It is used by the makefile system, but not nightly.
.RE
.LP
.B ONBLD_TOOLS
.RS 5
ONBLD_TOOLS is the root of all the tools that are part of SUNWonbld; e.g.,
/ws/onnv-tools/onbld.  By default, it is derived from
.BR BUILD_TOOLS .
It is used by the makefile system, but not nightly.
.RE
.LP
.B SPRO_ROOT
.RS 5
The gate-defined default location for the Sun compilers, e.g.
/ws/onnv-tools/SUNWspro.  By default, it is derived from
.BR BUILD_TOOLS .
It is used by the makefile system, but not nightly.
.RE
.LP
.B JAVA_ROOT
.RS 5
The location for the java compilers for the build, generally /usr/java.
.RE
.LP
.B OPTHOME
.RS 5
The gate-defined default location of things formerly in /opt; e.g.,
/ws/onnv-tools.  This is used by nightly, but not the makefiles.
.RE
.LP
.B TEAMWARE
.RS 5
The gate-defined default location for the Teamware tools; e.g.,
/ws/onnv-tools/SUNWspro.  By default, it is derived from
.BR OPTHOME .
This is used by nightly, but not the makefiles.  There is no
corresponding variable for Mercurial or Subversion, which are assumed
to be installed in the default path.
.RE
.LP
.B ON_CLOSED_BINS
.RS 5
OpenSolaris builds do not contain the closed source tree.  Instead,
the developer downloads a closed binaries tree and unpacks it.
.B ON_CLOSED_BINS
tells nightly
where to find these closed binaries, so that it can add them into the
build.
.LP
.RE
.B ON_CRYPTO_BINS
.RS 5
This is the path to a compressed tarball that contains debug
cryptographic binaries that have been signed to allow execution
outside of Sun, e.g., $PARENT_WS/packages/$MACH/on-crypto.$MACH.bz2.
.I nightly
will automatically adjust the path for non-debug builds.  This tarball
is needed if the closed-source tree is not present.  Also, it is
usually needed when generating OpenSolaris deliverables from a project
workspace.  This is because most projects do not have access to the
necessary key and certificate that would let them sign their own
cryptographic binaries.
.LP
.RE
.B CHECK_PATHS
```

```
.RS 5
Normally, nightly runs the 'checkpaths' script to check for
discrepancies among the files that list paths to other files, such as
exception lists and req.flg.  Set this flag to 'n' to disable this
check, which appears in the nightly output as "Check lists of files."
.RE
.LP
.B CHECK_DMAKE
.RS 5
Nightly validates that the version of dmake encountered is known to be
safe to use.  Set this flag to 'n' to disable this test, allowing any
version of dmake to be used.
.RE
.LP
.B MULTI_PROTO
.RS 5
If "no" (the default),
.I nightly
will reuse $ROOT for both the DEBUG and non-DEBUG builds.  If "yes",
the DEBUG build will go in $ROOT and the non-DEBUG build will go in
$ROOT-nd.  Other values will be treated as "no".
.RE
.LP
.SH NIGHTLY HOOK ENVIRONMENT VARIABLES
.LP
Several optional environment variables may specify commands to run at
various points during the build.  Commands specified in the hook
variable will be run in a subshell; command output will be appended to
the mail message and log file.  If the hook exits with a non-zero
status, the build is aborted immediately.  Environment variables
defined in the environment file will be available.
.LP
.B SYS_PRE_NIGHTLY
.RS 5
Run just after the workspace lock is acquired.  This is reserved for
per-build-machine customizations and should be set only in /etc/nightly.conf
.RE
.LP
.B PRE_NIGHTLY
.RS 5
Run just after SYS_PRE_NIGHTLY.
.RE
.LP
.B PRE_BRINGOVER
.RS 5
Run just before bringover is started; not run if no bringover is done.
.RE
.LP
.B POST_BRINGOVER
.RS 5
Run just after bringover completes; not run if no bringover is done.
.RE
.LP
.B POST_NIGHTLY
.RS 5
Run after the build completes, with the return status of nightly - one
of "Completed", "Interrupted", or "Failed" - available in the
environment variable NIGHTLY_STATUS.
.RE
.LP
.B SYS_POST_NIGHTLY
.RS 5
This is reserved for per-build-machine customizations, and runs
immedately after POST_NIGHTLY.
.RE
.LP
```

```
.SH REALMODE ENVIRONMENT VARIABLES
.LP
The following environment variables referenced by
.I nightly
are only required when the -X option is used.
.LP
.RE
.B IA32_IHV_WS
.RS 5
Reference to the IHV workspace containing IHV driver binaries.
The IHV workspace must be fully built before starting the ON realmode build.
.LP
.RE
.B IA32_IHV_ROOT
.RS 5
Reference to the IHV workspace proto area.
The IHV workspace must be fully built before starting the ON realmode build.
.LP
.RE
.B IA32_IHV_PKGS
.RS 5
Reference to the IHV workspace packages.  If this is empty or the directory
is non-existent, then nightly will skip copying the packages.
.LP
.RE
.B IA32_IHV_BINARY_PKGS
.RS 5
Reference to binary-only IHV packages.  If this is empty or the directory
is non-existent, then nightly will skip copying the packages.
.LP
.RE
.B SPARC_RM_PKGARCHIVE
.RS 5
Destination for sparc realmode package SUNWrmodu.
Yes, this sparc package really is built on x86.
.SH FILES
.LP
.RS 5
/etc/nightly.conf
.RE
.LP
If present, nightly executes this file just prior to executing the
.I env
file.
.SH BUILDING THE IHV WORKSPACE
.LP
The IHV workspace can be built with
.I nightly.
The recommended options are:
.LP
.RS 5
NIGHTLY_OPTIONS="-pmWN"
.RE
.LP
None of the realmode environment variables needed for ON realmode builds
are required to build the IHV workspace.
.SH EXAMPLES
.LP
Start with the example file in usr/src/tools/env/developer.sh
(or gatekeeper.sh), copy to myenv and make your changes.
.LP
.PD 0
# grep NIGHTLY_OPTIONS myenv
.LP
NIGHTLY_OPTIONS="-ACrlapDm"
.LP
```

```
export NIGHTLY_OPTIONS
.LP
# /opt/onbld/bin/nightly -i myenv
.PD
.LP
.SH SEE ALSO
.BR bldenv (1)
```

```
**********************************************************
   59019 Sun Jan 26 22:00:01 2014
new/usr/src/tools/scripts/nightly.sh
4526 nightly contains a great deal of effectively dead code
**********************************************************
#!/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 (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2008, 2010, Richard Lowe
# Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
# Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
#
# Based on the nightly script from the integration folks,
# Mostly modified and owned by mike_s.
# Changes also by kjc, dmk.
#
# BRINGOVER_WS may be specified in the env file.
# The default is the old behavior of CLONE_WS
#
# -i on the command line, means fast options, so when it's on the
# command line (only), lint and check builds are skipped no matter what
# the setting of their individual flags are in NIGHTLY_OPTIONS.
#
# LINTDIRS can be set in the env file, format is a list of:
#
#       /dirname-to-run-lint-on flag
#
#       Where flag is:  y - enable lint noise diff output
#                       n - disable lint noise diff output
#
#       For example: LINTDIRS="$SRC/uts n $SRC/stand y $SRC/psm y"
#
# OPTHOME  may be set in the environment to override /opt
# OPTHOME and TEAMWARE may be set in the environment to override /opt
# and /opt/teamware defaults.
#

#
# The CDPATH variable causes ksh's 'cd' builtin to emit messages to stdout
# under certain circumstances, which can really screw things up; unset it.
#
unset CDPATH

# Get the absolute path of the nightly script that the user invoked.  This
# may be a relative path, and we need to do this before changing directory.
```

```
nightly_path='whence $0'

#
# Keep track of where we found nightly so we can invoke the matching
# which_scm script.  If that doesn't work, don't go guessing, just rely
# on the $PATH settings, which will generally give us either /opt/onbld
# or the user's workspace.
#
WHICH_SCM=$(dirname $nightly_path)/which_scm
if [[ ! -x $WHICH_SCM ]]; then
        WHICH_SCM=which_scm
fi

function fatal_error
{
        print -u2 "nightly: $*"
        exit 1
}

#
# Function to do a DEBUG and non-DEBUG build. Needed because we might
# need to do another for the source build, and since we only deliver DEBUG or
# non-DEBUG packages.
#
# usage: normal_build
#
function normal_build {

        typeset orig_p_FLAG="$p_FLAG"
        typeset crypto_signer="$CODESIGN_USER"

        suffix=""

        # non-DEBUG build begins

        if [ "$F_FLAG" = "n" ]; then
                set_non_debug_build_flags
                CODESIGN_USER="$crypto_signer" \
                    build "non-DEBUG" "$suffix-nd" "-nd" "$MULTI_PROTO"
                if [ "$build_ok" = "y" -a "$X_FLAG" = "y" -a \
                    "$p_FLAG" = "y" ]; then
                        copy_ihv_pkgs non-DEBUG -nd
                fi
        else
                echo "\n==== No non-DEBUG $open_only build ====\n" >> "$LOGFILE"
        fi

        # non-DEBUG build ends

        # DEBUG build begins

        if [ "$D_FLAG" = "y" ]; then
                set_debug_build_flags
                CODESIGN_USER="$crypto_signer" \
                    build "DEBUG" "$suffix" "" "$MULTI_PROTO"
                if [ "$build_ok" = "y" -a "$X_FLAG" = "y" -a \
                    "$p_FLAG" = "y" ]; then
                        copy_ihv_pkgs DEBUG ""
                fi
        else
                echo "\n==== No DEBUG $open_only build ====\n" >> "$LOGFILE"
        fi

        # DEBUG build ends

        p_FLAG="$orig_p_FLAG"
}
```
______**unchanged_portion_omitted_**

```
#
# usage: filelist DESTDIR PATTERN
#
function filelist {
        DEST=$1
        PATTERN=$2
        cd ${DEST}
}

# function to save off binaries after a full build for later
# restoration
function save_binaries {
        # save off list of binaries
        echo "\n==== Saving binaries from build at ‘date‘ ====\n" | \
            tee -a $mail_msg_file >> $LOGFILE
        rm -f ${BINARCHIVE}
        cd ${CODEMGR_WS}
        filelist ${CODEMGR_WS} ’^preserve:’ >> $LOGFILE
        filelist ${CODEMGR_WS} ’^preserve:’ | \
            cpio -ocB 2>/dev/null | compress \
            > ${BINARCHIVE}
}

# delete files
# usage: hybridize_files DESTDIR MAKE_TARGET
function hybridize_files {
        DEST=$1
        MAKETARG=$2

        echo "\n==== Hybridizing files at ‘date‘ ====\n" | \
            tee -a $mail_msg_file >> $LOGFILE
        for i in ‘filelist ${DEST} ’^delete:’‘
        do
                echo "removing ${i}." | tee -a $mail_msg_file >> $LOGFILE
                rm -rf "${i}"
        done
        for i in ‘filelist ${DEST} ’^hybridize:’ ‘
        do
                echo "hybridizing ${i}." | tee -a $mail_msg_file >> $LOGFILE
                rm -f ${i}+
                sed -e "/^# HYBRID DELETE START/,/^# HYBRID DELETE END/d" \
                    < ${i} > ${i}+
                mv ${i}+ ${i}
        done
}

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

        echo "\n==== Restoring binaries to ${MAKETARG} at ‘date‘ ====\n" | \
            tee -a $mail_msg_file >> $LOGFILE
        cd ${DEST}
        zcat ${BINARCHIVE} | \
            cpio -idmucvB 2>/dev/null | tee -a $mail_msg_file >> ${LOGFILE}
}

# rename files we save binaries of
# usage: rename_files DESTDIR MAKE_TARGET
function rename_files {
        DEST=$1
```

```
        MAKETARG=$2
        echo "\n==== Renaming source files in ${MAKETARG} at ‘date‘ ====\n" | \
            tee -a $mail_msg_file >> $LOGFILE
        for i in ‘filelist ${DEST} ’^rename:’‘
        do
                echo ${i} | tee -a $mail_msg_file >> ${LOGFILE}
                rm -f ${i}.export
                mv ${i} ${i}.export
        done
}

# Return library search directive as function of given root.
function myldlibs {
        echo "-L$1/lib -L$1/usr/lib"
}
```
______**unchanged_portion_omitted_**

```
#
# Function to do the build, including package generation.
# usage: build LABEL SUFFIX ND MULTIPROTO
# - LABEL is used to tag build output.
# - SUFFIX is used to distinguish files (e.g., DEBUG vs non-DEBUG,
#   open-only vs full tree).
# - ND is "-nd" (non-DEBUG builds) or "" (DEBUG builds).
# - If MULTIPROTO is "yes", it means to name the proto area according to
#   SUFFIX.  Otherwise ("no"), (re)use the standard proto area.
#
function build {
        LABEL=$1
        SUFFIX=$2
        ND=$3
        MULTIPROTO=$4
        INSTALLOG=install${SUFFIX}-${MACH}
        NOISE=noise${SUFFIX}-${MACH}
        PKGARCHIVE=${PKGARCHIVE_ORIG}${SUFFIX}

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

        export ENVLDLIBS1=‘myldlibs $ROOT‘
        export ENVCPPFLAGS1=‘myheaders $ROOT‘

        this_build_ok=y
        #
        #       Build OS-Networking source
        #
        echo "\n==== Building OS-Net source at ‘date‘ ($LABEL) ====\n" \
                >> $LOGFILE

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

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

        echo "\n==== Build errors ($LABEL) ====\n" >> $mail_msg_file
        egrep ":" $SRC/${INSTALLOG}.out |
                egrep -e "(^${MAKE}:|[  ]error[:        \n])" | \
                egrep -v "Ignoring unknown host" | \
                egrep -v "cc .* -o error " | \
                egrep -v "warning" >> $mail_msg_file
        if [ "$?" = "0" ]; then
                build_ok=n
                this_build_ok=n
        fi
        grep "bootblock image is .* bytes too big" $SRC/${INSTALLOG}.out \
                >> $mail_msg_file
        if [ "$?" = "0" ]; then
                build_ok=n
                this_build_ok=n
        fi

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

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

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

        if [ "$i_FLAG" = "n" ]; then
        if [ "$i_FLAG" = "n" -a "$W_FLAG" = "n" ]; then
                rm -f $SRC/${NOISE}.ref
                if [ -f $SRC/${NOISE}.out ]; then
                        mv $SRC/${NOISE}.out $SRC/${NOISE}.ref
                fi
                grep : $SRC/${INSTALLOG}.out \
                        | egrep -v '^/' \
                        | egrep -v '^(Start|Finish|real|user|sys|./bld_awk)' \
                        | egrep -v '^tic:' \
                        | egrep -v '^mcs' \
                        | egrep -v '^LD_LIBRARY_PATH=' \
                        | egrep -v 'ar: creating' \
                        | egrep -v 'ar: writing' \
                        | egrep -v 'conflicts:' \
                        | egrep -v ':saved created' \
                        | egrep -v '^stty.*c:' \
                        | egrep -v '^mfgname.c:' \
                        | egrep -v '^uname-i.c:' \
                        | egrep -v '^volumes.c:' \
                        | egrep -v '^lint library construction:' \
                        | egrep -v 'tsort: INFORM:' \
                        | egrep -v 'stripalign:' \
                        | egrep -v 'chars, width' \
                        | egrep -v "symbol (\'|')timezone' has differing types:"
                        | egrep -v 'PSTAMP' \
                        | egrep -v '|%WHOANDWHERE%|' \
                        | egrep -v '^Manifying' \
                        | egrep -v 'Ignoring unknown host' \
                        | egrep -v 'Processing method:' \
                        | egrep -v '^Writing' \
                        | egrep -v 'spellin1:' \
                        | egrep -v '^adding:' \
                        | egrep -v "^echo 'msgid" \
                        | egrep -v '^echo ' \
                        | egrep -v '\.c:$' \
                        | egrep -v '^Adding file:' \
                        | egrep -v 'CLASSPATH=' \
```

```
                        | egrep -v '\/var\/mail\/:saved' \
                        | egrep -v -- '-DUTS_VERSION=' \
                        | egrep -v '^Running Mkbootstrap' \
                        | egrep -v '^Applet length read:' \
                        | egrep -v 'bytes written:' \
                        | egrep -v '^File:SolarisAuthApplet.bin' \
                        | egrep -v -i 'jibversion' \
                        | egrep -v '^Output size:' \
                        | egrep -v '^Solo size statistics:' \
                        | egrep -v '^Using ROM API Version' \
                        | egrep -v '^Zero Signature length:' \
                        | egrep -v '^Note \(probably harmless\):' \
                        | egrep -v '::' \
                        | egrep -v -- '-xcache' \
                        | egrep -v '^\+' \
                        | egrep -v '^cc1: note: -fwritable-strings' \
                        | egrep -v 'svccfg-native -s svc:/' \
                        | sort | uniq >$SRC/${NOISE}.out
                if [ ! -f $SRC/${NOISE}.ref ]; then
                        cp $SRC/${NOISE}.out $SRC/${NOISE}.ref
                fi
                echo "\n==== Build noise differences ($LABEL) ====\n" \
                        >>$mail_msg_file
                diff $SRC/${NOISE}.ref $SRC/${NOISE}.out >>$mail_msg_file
        fi

        #
        #       Re-sign selected binaries using signing server
        #       (gatekeeper builds only)
        #
        if [ -n "$CODESIGN_USER" -a "$this_build_ok" = "y" ]; then
                echo "\n==== Signing proto area at `date` ====\n" >> $LOGFILE
                signing_file="${TMPDIR}/signing"
                rm -f ${signing_file}
                export CODESIGN_USER
                signproto $SRC/tools/codesign/creds 2>&1 | \
                        tee -a ${signing_file} >> $LOGFILE
                echo "\n==== Finished signing proto area at `date` ====\n" \
                    >> $LOGFILE
                echo "\n==== Crypto module signing errors ($LABEL) ====\n" \
                    >> $mail_msg_file
                egrep 'WARNING|ERROR' ${signing_file} >> $mail_msg_file
                if (( $? == 0 )) ; then
                        build_ok=n
                        this_build_ok=n
                fi
        fi

        #
        #       Building Packages
        #
        if [ "$p_FLAG" = "y" -a "$this_build_ok" = "y" ]; then
                if [ -d $SRC/pkg ]; then
                if [ -d $SRC/pkg -o -d $SRC/pkgdefs ]; then
                        echo "\n==== Creating $LABEL packages at `date` ====\n"
                                >> $LOGFILE
                        echo "Clearing out $PKGARCHIVE ..." >> $LOGFILE
                        rm -rf $PKGARCHIVE >> "$LOGFILE" 2>&1
                        mkdir -p $PKGARCHIVE >> "$LOGFILE" 2>&1

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

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

                        egrep "${MAKE}|ERROR|WARNING" $SRC/pkg/${INSTALLOG}.out
                        for d in pkg pkgdefs; do
                                if [ ! -f "$SRC/$d/Makefile" ]; then
                                        continue
                                fi

                                egrep "${MAKE}|ERROR|WARNING" $SRC/$d/${INSTALLO
                                grep ’:’ | \
                                grep -v PSTAMP | \
                                egrep -v "Ignoring unknown host" \
                                >> $mail_msg_file
                        done
                else
                        #
                        # Handle it gracefully if -p was set but there are
                        # neither pkg directories.
                        # neither pkg nor pkgdefs directories.
                        #
                        echo "\n==== No $LABEL packages to build ====\n" \
                                >> $LOGFILE
                fi
        else
                echo "\n==== Not creating $LABEL packages ====\n" >> $LOGFILE
        fi

        ROOT=$ORIGROOT
}
```
______unchanged_portion_omitted_

```
# Install proto area from IHV build

function copy_ihv_proto {

        echo "\n==== Installing IHV proto area ====\n" \
                >> $LOGFILE
        if [ -d "$IA32_IHV_ROOT" ]; then
                if [ ! -d "$ROOT" ]; then
                        echo "mkdir -p $ROOT" >> $LOGFILE
                        mkdir -p $ROOT
                fi
                echo "copying $IA32_IHV_ROOT to $ROOT\n" >> $LOGFILE
                cd $IA32_IHV_ROOT
                tar cf - . | (cd $ROOT; umask 0; tar xpf - ) 2>&1 >> $LOGFILE
        else
                echo "$IA32_IHV_ROOT: not found" >> $LOGFILE
        fi

        if [ "$MULTI_PROTO" = yes ]; then
                if [ ! -d "$ROOT-nd" ]; then
                        echo "mkdir -p $ROOT-nd" >> $LOGFILE
                        mkdir -p $ROOT-nd
                fi
                # If there’s a non-DEBUG version of the IHV proto area,
                # copy it, but copy something if there’s not.
                if [ -d "$IA32_IHV_ROOT-nd" ]; then
                        echo "copying $IA32_IHV_ROOT-nd to $ROOT-nd\n" >> $LOGFI
```

```
                        cd $IA32_IHV_ROOT-nd
                elif [ -d "$IA32_IHV_ROOT" ]; then
                        echo "copying $IA32_IHV_ROOT to $ROOT-nd\n" >> $LOGFILE
                        cd $IA32_IHV_ROOT
                else
                        echo "$IA32_IHV_ROOT{-nd,}: not found" >> $LOGFILE
                        return
                fi
                tar cf - . | (cd $ROOT-nd; umask 0; tar xpf - ) 2>&1 >> $LOGFILE
        fi
}

# Install IHV packages in PKGARCHIVE
# usage: copy_ihv_pkgs LABEL SUFFIX
function copy_ihv_pkgs {
        LABEL=$1
        SUFFIX=$2
        # always use non-DEBUG IHV packages
        IA32_IHV_PKGS=${IA32_IHV_PKGS_ORIG}-nd
        PKGARCHIVE=${PKGARCHIVE_ORIG}${SUFFIX}

        echo "\n==== Installing IHV packages from $IA32_IHV_PKGS ($LABEL) ====\n
                >> $LOGFILE
        if [ -d "$IA32_IHV_PKGS" ]; then
                cd $IA32_IHV_PKGS
                tar cf - * | \
                   (cd $PKGARCHIVE; umask 0; tar xpf - ) 2>&1 >> $LOGFILE
        else
                echo "$IA32_IHV_PKGS: not found" >> $LOGFILE
        fi

        echo "\n==== Installing IHV packages from $IA32_IHV_BINARY_PKGS ($LABEL)
                >> $LOGFILE
        if [ -d "$IA32_IHV_BINARY_PKGS" ]; then
                cd $IA32_IHV_BINARY_PKGS
                tar cf - * | \
                    (cd $PKGARCHIVE; umask 0; tar xpf - ) 2>&1 >> $LOGFILE
        else
                echo "$IA32_IHV_BINARY_PKGS: not found" >> $LOGFILE
        fi
}

#
# Build and install the onbld tools.
#
# usage: build_tools DESTROOT
#
# returns non-zero status if the build was successful.
#
function build_tools {
        DESTROOT=$1

        INSTALLOG=install-${MACH}

        echo "\n==== Building tools at ‘date‘ ====\n" \
                >> $LOGFILE

        rm -f ${TOOLS}/${INSTALLOG}.out
        cd ${TOOLS}
        /bin/time $MAKE TOOLS_PROTO=${DESTROOT} -e install 2>&1 | \
            tee -a ${TOOLS}/${INSTALLOG}.out >> $LOGFILE

        echo "\n==== Tools build errors ====\n" >> $mail_msg_file

        egrep ":" ${TOOLS}/${INSTALLOG}.out |
                egrep -e "(${MAKE}:|[   ]error[:        \n])" | \
                egrep -v "Ignoring unknown host" | \
                egrep -v warning >> $mail_msg_file
        return $?
}
______unchanged_portion_omitted_

#
# Verify that the closed bins are present
# Verify that the closed tree is present if it needs to be.
#
function check_closed_bins {
function check_closed_tree {
        if [[ ! -d "$ON_CLOSED_BINS" ]]; then
                echo "ON_CLOSED_BINS must point to the closed binaries tree."
                build_ok=n
                exit 1
        fi
}

function obsolete_build {
        echo "WARNING: Obsolete $1 build requested; request will be ignored"
}

#
# wrapper over wsdiff.
# usage: do_wsdiff LABEL OLDPROTO NEWPROTO
#
function do_wsdiff {
        label=$1
        oldproto=$2
        newproto=$3

        wsdiff="wsdiff"
        [ "$t_FLAG" = y ] && wsdiff="wsdiff -t"

        echo "\n==== Getting object changes since last build at ‘date‘" \
            "($label) ====\n" | tee -a $LOGFILE >> $mail_msg_file
        $wsdiff -s -r ${TMPDIR}/wsdiff.results $oldproto $newproto 2>&1 | \
                    tee -a $LOGFILE >> $mail_msg_file
        echo "\n==== Object changes determined at ‘date‘ ($label) ====\n" | \
            tee -a $LOGFILE >> $mail_msg_file
}
______unchanged_portion_omitted_


MACH=‘uname -p‘

if [ "$OPTHOME" = "" ]; then
        OPTHOME=/opt
        export OPTHOME
fi
if [ "$TEAMWARE" = "" ]; then
        TEAMWARE=$OPTHOME/teamware
        export TEAMWARE
fi

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

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

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

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

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

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

# default values for low-level FLAGS; G I R are group FLAGS
A_FLAG=n
C_FLAG=n
D_FLAG=n
F_FLAG=n
f_FLAG=n
i_FLAG=n; i_CMD_LINE_FLAG=n
l_FLAG=n
M_FLAG=n
m_FLAG=n
N_FLAG=n
n_FLAG=n
o_FLAG=n
P_FLAG=n
p_FLAG=n
r_FLAG=n
T_FLAG=n
t_FLAG=y
U_FLAG=n
u_FLAG=n
V_FLAG=n
W_FLAG=n
w_FLAG=n
X_FLAG=n
#
XMOD_OPT=
#
build_ok=y

#
# examine arguments
#

OPTIND=1
while getopts +intV: FLAG
do
	case $FLAG in
	  i )	i_FLAG=y; i_CMD_LINE_FLAG=y
		;;
	  n )	n_FLAG=y
		;;
	 +t )	t_FLAG=n
		;;
	  V )	V_FLAG=y
		V_ARG="$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

check_closed_bins
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 +ABCDdFfGIilMmNnpRrtUuw FLAG $NIGHTLY_OPTIONS
while getopts +ABCDdFfGIilMmNnoPpRrTtUuWwXxz FLAG $NIGHTLY_OPTIONS
do
        case $FLAG in
          A )   A_FLAG=y
                ;;
          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
                ;;
          P )   P_FLAG=y
                ;; # obsolete
          p )   p_FLAG=y
                ;;
          R )   m_FLAG=y
                p_FLAG=y
                ;;
          r )   r_FLAG=y
                ;;
          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:$OPTHOME/SUNWspro/bin:/usr/bin:/usr/sbin:/usr/ucb"
PATH="$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"

PROTOCMPTERSE="protocmp.terse -gu"
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.
export POUND_SIGN RELEASE_DATE DEV_CM
# 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"
if [[ -z "$MAKE" ]]; then
        MAKE=dmake
elif [[ ! -x "$MAKE" ]]; then
        echo "\$MAKE is set to garbage in the environment"
        exit 1
fi
# 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, if you really want to use this version
anyway add the following to your environment to disable this check:
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 [ "${SUNWSPRO}" != "" ]; then
        PATH="${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

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
}
newdirlist=

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

# since this script assumes the build is from full source, it nullifies
# variables likely to have been set by a "ws" script; nullification
# confines the search space for headers and libraries to the proto area
# built from this immediate source.
ENVLDLIBS1=
ENVLDLIBS2=
ENVLDLIBS3=
ENVCPPFLAGS1=
ENVCPPFLAGS2=
ENVCPPFLAGS3=
ENVCPPFLAGS4=
PARENT_ROOT=

export ENVLDLIBS3 ENVCPPFLAGS1 ENVCPPFLAGS2 ENVCPPFLAGS3 ENVCPPFLAGS4 \
        ENVLDLIBS1 ENVLDLIBS2 PARENT_ROOT

PKGARCHIVE_ORIG=$PKGARCHIVE
IA32_IHV_PKGS_ORIG=$IA32_IHV_PKGS

#
# Juggle the logs and optionally send mail on completion.
#

function logshuffle {
        LLOG="$ATLOG/log.`date '+%F.%H:%M'`"
        if [ -f $LLOG -o -d $LLOG ]; then
                LLOG=$LLOG.$$
        fi
        mkdir $LLOG
        export LLOG

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

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

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

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

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

        exec >>$LOGFILE 2>&1
        if [ -s $build_noise_file ]; then
                echo "\n==== Nightly build noise ====\n" |
                    tee -a $LOGFILE >>$mail_msg_file
                cat $build_noise_file >>$LOGFILE
                cat $build_noise_file >>$mail_msg_file
                echo | tee -a $LOGFILE >>$mail_msg_file
        fi
        rm -f $build_noise_file

        case "$build_ok" in
                y)
                        state=Completed
                        ;;
                i)
                        state=Interrupted
                        ;;
                *)
                        state=Failed
                        ;;
        esac
        NIGHTLY_STATUS=$state
        export NIGHTLY_STATUS

        run_hook POST_NIGHTLY $state
        run_hook SYS_POST_NIGHTLY $state

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

        cat $build_time_file $build_environ_file $mail_msg_file \
            > ${LLOG}/mail_msg
        if [ "$m_FLAG" = "y" ]; then
                cat ${LLOG}/mail_msg | /usr/bin/mailx ${mailx_r} -s \
        "Nightly ${MACH} Build of `basename ${CODEMGR_WS}` ${state}." \
                        ${MAILTO}
        fi

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

        mv $LOGFILE $LLOG
}
______unchanged_portion_omitted_

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

```
#       $ATLOG/nightly.lock
#               - protects against multiple builds in same workspace
#       $PARENT_WS/usr/src/nightly.$MACH.lock
#               - protects against multiple 'u' copy-backs
#       $NIGHTLY_PARENT_ROOT/nightly.lock
#               - protects against multiple 'U' copy-backs
#
# Overriding ISUSER to 1 causes the lock to be created as root if the
# script is run as root.  The default is to create it as $STAFFER.
ISUSER=1 create_lock $ATLOG/nightly.lock "atloglockfile"
if [ "$u_FLAG" = "y" ]; then
        create_lock $PARENT_WS/usr/src/nightly.$MACH.lock "ulockfile"
fi
if [ "$U_FLAG" = "y" ]; then
        # NIGHTLY_PARENT_ROOT is written as root if script invoked as root.
        ISUSER=1 create_lock $NIGHTLY_PARENT_ROOT/nightly.lock "Ulockfile"
fi

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

#
# Create mail_msg_file
#
mail_msg_file="${TMPDIR}/mail_msg"
touch $mail_msg_file
build_time_file="${TMPDIR}/build_time"
build_environ_file="${TMPDIR}/build_environ"
touch $build_environ_file
#
#       Move old LOGFILE aside
#       ATLOG directory already made by 'create_lock' above
#
if [ -f $LOGFILE ]; then
        mv -f $LOGFILE ${LOGFILE}-
fi
#
#       Build OsNet source
#
START_DATE=`date`
SECONDS=0
echo "\n==== Nightly $maketype build started:   $START_DATE ====" \
    | tee -a $LOGFILE > $build_time_file

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

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

run_hook SYS_PRE_NIGHTLY
run_hook PRE_NIGHTLY

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

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

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

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

```
fi

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

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

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

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

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

```
        fi
fi

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

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

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

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

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

        CODEMGR_WS="$BRINGOVER_WS" "$WHICH_SCM" 2>/dev/null \
            | read scm_type junk
        if [[ -z "$scm_type" || "$scm_type" == unknown ]]; then
                # Probe BRINGOVER_WS to determine its type
                if [[ $BRINGOVER_WS == ssh://* ]]; then
                if [[ $BRINGOVER_WS == svn*://* ]]; then
                        scm_type="subversion"
                elif [[ $BRINGOVER_WS == file://* ]] &&
                    egrep -s "This is a Subversion repository" \
                    ${BRINGOVER_WS#file://}/README.txt 2> /dev/null; then
                        scm_type="subversion"
                elif [[ $BRINGOVER_WS == ssh://* ]]; then
                        scm_type="mercurial"
                elif [[ $BRINGOVER_WS == http://* ]] && \
                    wget -q -O- --save-headers "$BRINGOVER_WS/?cmd=heads" | \
                    egrep -s "application/mercurial" 2> /dev/null; then
                        scm_type="mercurial"
                elif svn info $BRINGOVER_WS > /dev/null 2>&1; then
                        scm_type="subversion"
                else
                        scm_type="none"
                fi
        fi

        # fold both unsupported and unrecognized results into "none"
        case "$scm_type" in
        mercurial)
        none|subversion|teamware|mercurial)
                ;;
        *)      scm_type=none
                ;;
        esac

        echo $scm_type
}

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

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

        case "$scm_type" in
        none|git|mercurial)
        none|subversion|git|teamware|mercurial)
                ;;
        *)      scm_type=none
                ;;
        esac

        echo $scm_type
}

SCM_TYPE=$(child_wstype)

#
#       Decide whether to clobber
#
if [ "$i_FLAG" = "n" -a -d "$SRC" ]; then
        echo "\n==== Make clobber at `date` ====\n" >> $LOGFILE

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

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

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

        if [[ "$t_FLAG" = "y" ]]; then
                echo "\n==== Make tools clobber at `date` ====\n" >> $LOGFILE
                cd ${TOOLS}
                rm -f ${TOOLS}/clobber-${MACH}.out
                $MAKE TOOLS_PROTO=$TOOLS_PROTO -ek clobber 2>&1 | \
                        tee -a ${TOOLS}/clobber-${MACH}.out >> $LOGFILE
                echo "\n==== Make tools clobber ERRORS ====\n" \
                        >> $mail_msg_file
                grep "$MAKE:" ${TOOLS}/clobber-${MACH}.out \
                        >> $mail_msg_file
                rm -rf ${TOOLS_PROTO}
```

```
                mkdir -p ${TOOLS_PROTO}
        fi

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

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

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

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

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

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

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

        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"

        #
        # 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
}

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

        if [[ ! -d $CODEMGR_WS/.svn ]]; then
                staffer svn checkout $BRINGOVER_WS $CODEMGR_WS ||
                        touch $TMPDIR/bringover_failed
        else
                typeset root
                root=$(staffer svn info $CODEMGR_WS |
                        nawk '/^Repository Root:/ {print $NF}’)
                if [[ $root != $BRINGOVER_WS ]]; then
                        # We fail here because there’s no way to update
                        # from a named repo.
                        cat <<-EOF
                        \$BRINGOVER_WS doesn’t match repository root:
                          \$BRINGOVER_WS:  $BRINGOVER_WS
                          Repository root: $root
                        EOF
                        touch $TMPDIR/bringover_failed
                else
```

```
                        # If a conflict happens, svn still exits 0.
                        staffer svn update $CODEMGR_WS | tee $TMPDIR/pull.out ||
                                touch $TMPDIR/bringover_failed
                        if grep "^C" $TMPDIR/pull.out > /dev/null 2>&1; then
                                touch $TMPDIR/bringover_failed
                        fi
                fi
        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

        check_closed_bins
        check_closed_tree

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

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

normal_build

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


#
# 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.
#
```

```
# 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
```