```
************************************************************
   91520 Wed Dec 25 08:52:20 2013
new/usr/src/tools/scripts/webrev.sh
allow webrev to be run directly from usr/src/tools/scripts/webrev
************************************************************
#!/usr/bin/ksh93 -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) 2002, 2010, Oracle and/or its affiliates. All rights reserved.
#

# Copyright 2008, 2010, Richard Lowe
# Copyright 2012 Marcel Telka <marcel@telka.sk>
# Copyright 2013 PALO, Richard. All rights reserved.

#
# This script takes a file list and a workspace and builds a set of html files
# suitable for doing a code review of source changes via a web page.
# Documentation is available via the manual page, webrev.1, or just
# type 'webrev -h'.
#
# Acknowledgements to contributors to webrev are listed in the webrev(1)
# man page.
#

REMOVED_COLOR=brown
CHANGED_COLOR=blue
NEW_COLOR=blue

HTML='<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">\n'

FRAMEHTML='<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">\n'

STDHEAD='<meta http-equiv="cache-control" content="no-cache"></meta>
<meta http-equiv="Pragma" content="no-cache"></meta>
<meta http-equiv="Expires" content="-1"></meta>
<!--
   Note to customizers: the body of the webrev is IDed as SUNWwebrev
   to allow easy overriding by users of webrev via the userContent.css
   mechanism available in some browsers.
```

```
   For example, to have all "removed" information be red instead of
   brown, set a rule in your userContent.css file like:

       body#SUNWwebrev span.removed { color: red ! important; }
-->
<style type="text/css" media="screen">
body {
    background-color: #eeeeee;
}
______unchanged_portion_omitted_


#
# fix_postscript
#

#
#
# Main program starts here
#
#

trap "rm -f /tmp/$$.* ; exit" 0 1 2 3 15

set +o noclobber

PATH=$(/bin/dirname "$(whence $0)"):$PATH

[[ -z $WDIFF ]] && WDIFF=`look_for_prog wdiff`
[[ -z $WX ]] && WX=`look_for_prog wx`
[[ -z $HG_ACTIVE ]] && HG_ACTIVE=`look_for_prog hg-active`
[[ -z $GIT ]] && GIT=`look_for_prog git`
[[ -z $WHICH_SCM ]] && WHICH_SCM=`look_for_prog which_scm`
[[ -z $CODEREVIEW ]] && CODEREVIEW=`look_for_prog codereview`
[[ -z $PS2PDF ]] && PS2PDF=`look_for_prog ps2pdf`
[[ -z $PERL ]] && PERL=`look_for_prog perl`
[[ -z $RSYNC ]] && RSYNC=`look_for_prog rsync`
[[ -z $SCCS ]] && SCCS=`look_for_prog sccs`
[[ -z $AWK ]] && AWK=`look_for_prog nawk`
[[ -z $AWK ]] && AWK=`look_for_prog gawk`
[[ -z $AWK ]] && AWK=`look_for_prog awk`
[[ -z $SCP ]] && SCP=`look_for_prog scp`
[[ -z $SED ]] && SED=`look_for_prog sed`
[[ -z $SFTP ]] && SFTP=`look_for_prog sftp`
[[ -z $SORT ]] && SORT=`look_for_prog sort`
[[ -z $MKTEMP ]] && MKTEMP=`look_for_prog mktemp`
[[ -z $GREP ]] && GREP=`look_for_prog grep`
[[ -z $FIND ]] && FIND=`look_for_prog find`

# set name of trash directory for remote webrev deletion
TRASH_DIR=".trash"
[[ -n $WEBREV_TRASH_DIR ]] && TRASH_DIR=$WEBREV_TRASH_DIR

if [[ ! -x $PERL ]]; then
        print -u2 "Error: No perl interpreter found.  Exiting."
        exit 1
fi

if [[ ! -x $WHICH_SCM ]]; then
        print -u2 "Error: Could not find which_scm.  Exiting."
        exit 1
fi

#
# These aren't fatal, but we want to note them to the user.
# We don’t warn on the absence of ’wx’ until later when we’ve
# determined that we actually need to try to invoke it.
#
[[ ! -x $CODEREVIEW ]] && print -u2 "WARNING: codereview(1) not found."
[[ ! -x $PS2PDF ]] && print -u2 "WARNING: ps2pdf(1) not found."
[[ ! -x $WDIFF ]] && print -u2 "WARNING: wdiff not found."

# Declare global total counters.
integer TOTL TINS TDEL TMOD TUNC

# default remote host for upload/delete
typeset -r DEFAULT_REMOTE_HOST="cr.opensolaris.org"
# prefixes for upload targets
typeset -r rsync_prefix="rsync://"
typeset -r ssh_prefix="ssh://"

Cflag=
Dflag=
flist_mode=
flist_file=
iflag=
Iflag=
lflag=
Nflag=
nflag=
Oflag=
oflag=
pflag=
tflag=
uflag=
Uflag=
wflag=
remote_target=

#
# NOTE: when adding/removing options it is necessary to sync the list
#       with usr/src/tools/onbld/hgext/cdm.py
#
while getopts "C:Di:I:lnNo:Op:t:Uw" opt
do
        case $opt in
        C)      Cflag=1
                ITSCONF=$OPTARG;;

        D)      Dflag=1;;

        i)      iflag=1
                INCLUDE_FILE=$OPTARG;;

        I)      Iflag=1
                ITSREG=$OPTARG;;

        #
        # If -l has been specified, we need to abort further options
        # processing, because subsequent arguments are going to be
        # arguments to ’putback -n’.
        #
        l)      lflag=1
                break;;

        N)      Nflag=1;;

        n)      nflag=1;;

        O)      Oflag=1;;
```

```
        o)      oflag=1
                # Strip the trailing slash to correctly form remote target.
                WDIR=${OPTARG%/};;

        p)      pflag=1
                codemgr_parent=$OPTARG;;

        t)      tflag=1
                remote_target=$OPTARG;;

        U)      Uflag=1;;

        w)      wflag=1;;

        ?)      usage;;
        esac
done

FLIST=/tmp/$$.flist

if [[ -n $wflag && -n $lflag ]]; then
        usage
fi

# more sanity checking
if [[ -n $nflag && -z $Uflag ]]; then
        print "it does not make sense to skip webrev generation" \
            "without -U"
        exit 1
fi

if [[ -n $tflag && -z $Uflag && -z $Dflag ]]; then
        echo "remote target has to be used only for upload or delete"
        exit 1
fi

#
# For the invocation "webrev -n -U" with no other options, webrev will assume
# that the webrev exists in ${CWS}/webrev, but will upload it using the name
# $(basename ${CWS}).  So we need to get CWS set before we skip any remaining
# logic.
#
$WHICH_SCM | read SCM_MODE junk || exit 1
if [[ $SCM_MODE == "teamware" ]]; then
        #
        # Teamware priorities:
        # 1. CODEMGR_WS from the environment
        # 2. workspace name
        #
        [[ -z $codemgr_ws && -n $CODEMGR_WS ]] && codemgr_ws=$CODEMGR_WS
        if [[ -n $codemgr_ws && ! -d $codemgr_ws ]]; then
                print -u2 "$codemgr_ws: no such workspace"
                exit 1
        fi
        [[ -z $codemgr_ws ]] && codemgr_ws=$(workspace name)
        codemgr_ws=$(cd $codemgr_ws;print $PWD)
        CODEMGR_WS=$codemgr_ws
        CWS=$codemgr_ws
elif [[ $SCM_MODE == "mercurial" ]]; then
        #
        # Mercurial priorities:
        # 1. hg root from CODEMGR_WS environment variable
        # 1a. hg root from CODEMGR_WS/usr/closed if we’re somewhere under
        #    usr/closed when we run webrev
        # 2. hg root from directory of invocation
        #
        if [[ ${PWD} =~ "usr/closed" ]]; then
                testparent=${CODEMGR_WS}/usr/closed
                # If we're in OpenSolaris mode, we enforce a minor policy:
                # help to make sure the reviewer doesn't accidentally publish
                # source which is under usr/closed
                if [[ -n "$Oflag" ]]; then
                        print -u2 "OpenSolaris output not permitted with" \
                            "usr/closed changes"
                        exit 1
                fi
        else
                testparent=${CODEMGR_WS}
        fi
        [[ -z $codemgr_ws && -n $testparent ]] && \
            codemgr_ws=$(hg root -R $testparent 2>/dev/null)
        [[ -z $codemgr_ws ]] && codemgr_ws=$(hg root 2>/dev/null)
        CWS=$codemgr_ws
elif [[ $SCM_MODE == "git" ]]; then
        #
        # Git priorities:
        # 1. git rev-parse --git-dir from CODEMGR_WS environment variable
        # 2. git rev-parse --git-dir from directory of invocation
        #
        [[ -z $codemgr_ws && -n $CODEMGR_WS ]] && \
            codemgr_ws=$($GIT --git-dir=$CODEMGR_WS/.git rev-parse --git-dir \
                2>/dev/null)
        [[ -z $codemgr_ws ]] && \
            codemgr_ws=$($GIT rev-parse --git-dir 2>/dev/null)

        if [[ "$codemgr_ws" == ".git" ]]; then
                codemgr_ws="${PWD}/${codemgr_ws}"
        fi

        codemgr_ws=$(dirname $codemgr_ws) # Lose the '/.git'
        CWS="$codemgr_ws"
elif [[ $SCM_MODE == "subversion" ]]; then
        #
        # Subversion priorities:
        # 1. CODEMGR_WS from environment
        # 2. Relative path from current directory to SVN repository root
        #
        if [[ -n $CODEMGR_WS && -d $CODEMGR_WS/.svn ]]; then
                CWS=$CODEMGR_WS
        else
                svn info | while read line; do
                        if [[ $line == "URL: "* ]]; then
                                url=${line#URL: }
                        elif [[ $line == "Repository Root: "* ]]; then
                                repo=${line#Repository Root: }
                        fi
                done

                rel=${url#$repo}
                CWS=${PWD%$rel}
        fi
fi

#
# If no SCM has been determined, take either the environment setting
# setting for CODEMGR_WS, or the current directory if that wasn't set.
#
if [[ -z ${CWS} ]]; then
        CWS=${CODEMGR_WS:-.}
fi

#
```

```
# If the command line options indicate no webrev generation, either
# explicitly (-n) or implicitly (-D but not -U), then there's a whole
# ton of logic we can skip.
#
# Instead of increasing indentation, we intentionally leave this loop
# body open here, and exit via break from multiple points within.
# Search for DO_EVERYTHING below to find the break points and closure.
#
for do_everything in 1; do

# DO_EVERYTHING: break point
if [[ -n $nflag || ( -z $Uflag && -n $Dflag ) ]]; then
        break
fi

#
# If this manually set as the parent, and it appears to be an earlier webrev,
# then note that fact and set the parent to the raw_files/new subdirectory.
#
if [[ -n $pflag && -d $codemgr_parent/raw_files/new ]]; then
        parent_webrev=$(readlink -f "$codemgr_parent")
        codemgr_parent=$(readlink -f "$codemgr_parent/raw_files/new")
fi

if [[ -z $wflag && -z $lflag ]]; then
        shift $(($OPTIND - 1))

        if [[ $1 == "-" ]]; then
                cat > $FLIST
                flist_mode="stdin"
                flist_done=1
                shift
        elif [[ -n $1 ]]; then
                if [[ ! -r $1 ]]; then
                        print -u2 "$1: no such file or not readable"
                        usage
                fi
                cat $1 > $FLIST
                flist_mode="file"
                flist_file=$1
                flist_done=1
                shift
        else
                flist_mode="auto"
        fi
fi

#
# Before we go on to further consider -l and -w, work out which SCM we think
# is in use.
#
case "$SCM_MODE" in
teamware|mercurial|git|subversion)
        ;;
unknown)
        if [[ $flist_mode == "auto" ]]; then
                print -u2 "Unable to determine SCM in use and file list not spec
                print -u2 "See which_scm(1) for SCM detection information."
                exit 1
        fi
        ;;
*)
        if [[ $flist_mode == "auto" ]]; then
                print -u2 "Unsupported SCM in use ($SCM_MODE) and file list not
                exit 1
        fi
        ;;
esac

print -u2 "   SCM detected: $SCM_MODE"

if [[ -n $lflag ]]; then
        #
        # If the -l flag is given instead of the name of a file list,
        # then generate the file list by extracting file names from a
        # putback -n.
        #
        shift $(($OPTIND - 1))
        if [[ $SCM_MODE == "teamware" ]]; then
                flist_from_teamware "$*"
        else
                print -u2 -- "Error: -l option only applies to TeamWare"
                exit 1
        fi
        flist_done=1
        shift $#
elif [[ -n $wflag ]]; then
        #
        # If the -w is given then assume the file list is in Bonwick's "wx"
        # command format, i.e.  pathname lines alternating with SCCS comment
        # lines with blank lines as separators.  Use the SCCS comments later
        # in building the index.html file.
        #
        shift $(($OPTIND - 1))
        wxfile=$1
        if [[ -z $wxfile && -n $CODEMGR_WS ]]; then
                if [[ -r $CODEMGR_WS/wx/active ]]; then
                        wxfile=$CODEMGR_WS/wx/active
                fi
        fi

        [[ -z $wxfile ]] && print -u2 "wx file not specified, and could not " \
            "be auto-detected (check \$CODEMGR_WS)" && exit 1

        if [[ ! -r $wxfile ]]; then
                print -u2 "$wxfile: no such file or not readable"
                usage
        fi

        print -u2 " File list from: wx 'active' file '$wxfile' ... \c"
        flist_from_wx $wxfile
        flist_done=1
        if [[ -n "$*" ]]; then
                shift
        fi
elif [[ $flist_mode == "stdin" ]]; then
        print -u2 " File list from: standard input"
elif [[ $flist_mode == "file" ]]; then
        print -u2 " File list from: $flist_file"
fi

if [[ $# -gt 0 ]]; then
        print -u2 "WARNING: unused arguments: $*"
fi

#
# Before we entered the DO_EVERYTHING loop, we should have already set CWS
# and CODEMGR_WS as needed.  Here, we set the parent workspace.
#

if [[ $SCM_MODE == "teamware" ]]; then
```

```
        #
        # Teamware priorities:
        #
        #      1) via -p command line option
        #      2) in the user environment
        #      3) in the flist
        #      4) automatically based on the workspace
        #

        #
        # For 1, codemgr_parent will already be set.  Here's 2:
        #
        [[ -z $codemgr_parent && -n $CODEMGR_PARENT ]] && \
            codemgr_parent=$CODEMGR_PARENT
        if [[ -n $codemgr_parent && ! -d $codemgr_parent ]]; then
                print -u2 "$codemgr_parent: no such directory"
                exit 1
        fi

        #
        # If we're in auto-detect mode and we haven't already gotten the file
        # list, then see if we can get it by probing for wx.
        #
        if [[ -z $flist_done && $flist_mode == "auto" && -n $codemgr_ws ]]; then
                if [[ ! -x $WX ]]; then
                        print -u2 "WARNING: wx not found!"
                fi

                #
                # We need to use wx list -w so that we get renamed files, etc.
                # but only if a wx active file exists-- otherwise wx will
                # hang asking us to initialize our wx information.
                #
                if [[ -x $WX && -f $codemgr_ws/wx/active ]]; then
                        print -u2 " File list from: 'wx list -w' ... \c"
                        $WX list -w > $FLIST
                        $WX comments > /tmp/$$.wx_comments
                        wxfile=/tmp/$$.wx_comments
                        print -u2 "done"
                        flist_done=1
                fi
        fi

        #
        # If by hook or by crook we've gotten a file list by now (perhaps
        # from the command line), eval it to extract environment variables from
        # it: This is method 3 for finding the parent.
        #
        if [[ -z $flist_done ]]; then
                flist_from_teamware
        fi
        env_from_flist

        #
        # (4) If we still don't have a value for codemgr_parent, get it
        # from workspace.
        #
        [[ -z $codemgr_parent ]] && codemgr_parent=`workspace parent`
        if [[ ! -d $codemgr_parent ]]; then
                print -u2 "$CODEMGR_PARENT: no such parent workspace"
                exit 1
        fi

        PWS=$codemgr_parent

        [[ -n $parent_webrev ]] && RWS=$(workspace parent $CWS)
```

```
elif [[ $SCM_MODE == "mercurial" ]]; then
        #
        # Parent can either be specified with -p
        # Specified with CODEMGR_PARENT in the environment
        # or taken from hg’s default path.
        #

        if [[ -z $codemgr_parent && -n $CODEMGR_PARENT ]]; then
                codemgr_parent=$CODEMGR_PARENT
        fi

        if [[ -z $codemgr_parent ]]; then
                codemgr_parent=‘hg path -R $codemgr_ws default 2>/dev/null‘
        fi

        PWS=$codemgr_parent

        #
        # If the parent is a webrev, we want to do some things against
        # the natural workspace parent (file list, comments, etc)
        #
        if [[ -n $parent_webrev ]]; then
                real_parent=$(hg path -R $codemgr_ws default 2>/dev/null)
        else
                real_parent=$PWS
        fi

        #
        # If hg-active exists, then we run it.  In the case of no explicit
        # flist given, we’ll use it for our comments.  In the case of an
        # explicit flist given we’ll try to use it for comments for any
        # files mentioned in the flist.
        #
        if [[ -z $flist_done ]]; then
                flist_from_mercurial $CWS $real_parent
                flist_done=1
        fi

        #
        # If we have a file list now, pull out any variables set
        # therein.  We do this now (rather than when we possibly use
        # hg-active to find comments) to avoid stomping specifications
        # in the user-specified flist.
        #
        if [[ -n $flist_done ]]; then
                env_from_flist
        fi

        #
        # Only call hg-active if we don’t have a wx formatted file already
        #
        if [[ -x $HG_ACTIVE && -z $wxfile ]]; then
                print "  Comments from: hg-active -p $real_parent ...\c"
                hg_active_wxfile $CWS $real_parent
                print " Done."
        fi

        #
        # At this point we must have a wx flist either from hg-active,
        # or in general.  Use it to try and find our parent revision,
        # if we don’t have one.
        #
        if [[ -z $HG_PARENT ]]; then
                eval ‘$SED -e "s/#.*$//" $wxfile | $GREP HG_PARENT=‘
        fi
```

```
        #
        # If we still don’t have a parent, we must have been given a
        # wx-style active list with no HG_PARENT specification, run
        # hg-active and pull an HG_PARENT out of it, ignore the rest.
        #
        if [[ -z $HG_PARENT && -x $HG_ACTIVE ]]; then
                $HG_ACTIVE -w $codemgr_ws -p $real_parent | \
                    eval ‘$SED -e "s/#.*$//" | $GREP HG_PARENT=‘
        elif [[ -z $HG_PARENT ]]; then
                print -u2 "Error: Cannot discover parent revision"
                exit 1
        fi

        pnode=$(trim_digest $HG_PARENT)
        PRETTY_PWS="${PWS} (at ${pnode})"
        cnode=$(hg parent -R $codemgr_ws --template ’{node|short}’ \
            2>/dev/null)
        PRETTY_CWS="${CWS} (at ${cnode})"}
elif [[ $SCM_MODE == "git" ]]; then
        #
        # Parent can either be specified with -p, or specified with
        # CODEMGR_PARENT in the environment.
        #

        if [[ -z $codemgr_parent && -n $CODEMGR_PARENT ]]; then
                codemgr_parent=$CODEMGR_PARENT
        fi

        # Try to figure out the parent based on the branch the current
        # branch is tracking, if we fail, use origin/master
        this_branch=$($GIT branch | nawk ’$1 == "*" { print $2 }’)
        par_branch="origin/master"

        # If we’re not on a branch there’s nothing we can do
        if [[ $this_branch != "(no branch)" ]]; then
                $GIT for-each-ref                                               \
                    --format=’%(refname:short) %(upstream:short)’ refs/heads/ |
                    while read local remote; do
                        [[ "$local" == "$this_branch" ]] && par_branch="$remote"
                    done
        fi

        if [[ -z $codemgr_parent ]]; then
                codemgr_parent=$par_branch
        fi
        PWS=$codemgr_parent

        #
        # If the parent is a webrev, we want to do some things against
        # the natural workspace parent (file list, comments, etc)
        #
        if [[ -n $parent_webrev ]]; then
                real_parent=$par_branch
        else
                real_parent=$PWS
        fi

        if [[ -z $flist_done ]]; then
                flist_from_git "$CWS" "$real_parent"
                flist_done=1
        fi

        #
        # If we have a file list now, pull out any variables set
        # therein.
        #
        if [[ -n $flist_done ]]; then
                env_from_flist
        fi

        #
        # If we don’t have a wx-format file list, build one we can pull change
        # comments from.
        #
        if [[ -z $wxfile ]]; then
                print "  Comments from: git...\c"
                git_wxfile "$CWS" "$real_parent"
                print " Done."
        fi

        if [[ -z $GIT_PARENT ]]; then
                GIT_PARENT=$($GIT merge-base "$real_parent" HEAD)
        fi
        if [[ -z $GIT_PARENT ]]; then
                print -u2 "Error: Cannot discover parent revision"
                exit 1
        fi

        pnode=$(trim_digest $GIT_PARENT)

        if [[ $real_parent == */* ]]; then
                origin=$(echo $real_parent | cut -d/ -f1)
                origin=$($GIT remote -v | \
                    $AWK ’$1 == "’$origin’" { print $2; exit }’)
                PRETTY_PWS="${PWS} (${origin} at ${pnode})"
        else
                PRETTY_PWS="${PWS} (at ${pnode})"
        fi

        cnode=$($GIT --git-dir=${codemgr_ws}/.git rev-parse --short=12 HEAD \
            2>/dev/null)
        PRETTY_CWS="${CWS} (at ${cnode})"
elif [[ $SCM_MODE == "subversion" ]]; then

        #
        # We only will have a real parent workspace in the case one
        # was specified (be it an older webrev, or another checkout).
        #
        [[ -n $codemgr_parent ]] && PWS=$codemgr_parent

        if [[ -z $flist_done && $flist_mode == "auto" ]]; then
                flist_from_subversion $CWS $OLDPWD
        fi
else
    if [[ $SCM_MODE == "unknown" ]]; then
        print -u2 "    Unknown type of SCM in use"
    else
        print -u2 "    Unsupported SCM in use: $SCM_MODE"
    fi

    env_from_flist

    if [[ -z $CODEMGR_WS ]]; then
        print -u2 "SCM not detected/supported and CODEMGR_WS not specified"
        exit 1
    fi

    if [[ -z $CODEMGR_PARENT ]]; then
        print -u2 "SCM not detected/supported and CODEMGR_PARENT not specified"
        exit 1
    fi
```

```
    CWS=$CODEMGR_WS
    PWS=$CODEMGR_PARENT
fi

#
# If the user didn’t specify a -i option, check to see if there is a
# webrev-info file in the workspace directory.
#
if [[ -z $iflag && -r "$CWS/webrev-info" ]]; then
        iflag=1
        INCLUDE_FILE="$CWS/webrev-info"
fi

if [[ -n $iflag ]]; then
        if [[ ! -r $INCLUDE_FILE ]]; then
                print -u2 "include file ’$INCLUDE_FILE’ does not exist or is" \
                    "not readable."
                exit 1
        else
                #
                # $INCLUDE_FILE may be a relative path, and the script alters
                # PWD, so we just stash a copy in /tmp.
                #
                cp $INCLUDE_FILE /tmp/$$.include
        fi
fi

# DO_EVERYTHING: break point
if [[ -n $Nflag ]]; then
        break
fi

typeset -A itsinfo
typeset -r its_sed_script=/tmp/$$.its_sed
valid_prefixes=
if [[ -z $nflag ]]; then
        DEFREGFILE="$(/bin/dirname "$(whence $0)")/../etc/its.reg"
        DEFREGFILE0="$(/bin/dirname "$(whence $0)")/its.reg"
        if [[ -n $Iflag ]]; then
                REGFILE=$ITSREG
        elif [[ -r $HOME/.its.reg ]]; then
                REGFILE=$HOME/.its.reg
        elif [[ -r $DEFREGFILE ]]; then
                REGFILE=$DEFREGFILE
        else
                REGFILE=$DEFREGFILE0
                REGFILE=$DEFREGFILE
        fi
        if [[ ! -r $REGFILE ]]; then
                print "ERROR: Unable to read database registry file $REGFILE"
                exit 1
        elif [[ $REGFILE != $DEFREGFILE ]]; then
                print "   its.reg from: $REGFILE"
        fi

        $SED -e ’/^#/d’ -e ’/^[ 	]*$/d’ $REGFILE | while read LINE; do

                name=${LINE%%=*}
                value="${LINE#*=}"

                if [[ $name == PREFIX ]]; then
                        p=${value}
                        valid_prefixes="${p} ${valid_prefixes}"
                else
                        itsinfo["${p}_${name}"]="${value}"
                fi
        done


        DEFCONFFILE="$(/bin/dirname "$(whence $0)")/../etc/its.conf"
        DEFCONFFILE0="$(/bin/dirname "$(whence $0)")/its.conf"
        if [[ -r $DEFCONFFILE ]]; then
                CONFFILES=$DEFCONFFILE
        else
                CONFFILES=$DEFCONFFILE0
        fi
        if [[ -r $HOME/.its.conf ]]; then
                CONFFILES="${CONFFILES} $HOME/.its.conf"
        fi
        if [[ -n $Cflag ]]; then
                CONFFILES="${CONFFILES} ${ITSCONF}"
        fi
        its_domain=
        its_priority=
        for cf in ${CONFFILES}; do
                if [[ ! -r $cf ]]; then
                        print "ERROR: Unable to read database configuration file
                        exit 1
                elif [[ $cf != $DEFCONFFILE ]]; then
                        print "       its.conf: reading $cf"
                fi
                $SED -e '/^#/d' -e '/^[         ]*$/d' $cf | while read LINE; do
                    eval "${LINE}"
                done
        done

        #
        # If an information tracking system is explicitly identified by prefix,
        # we want to disregard the specified priorities and resolve it according
        #
        # To that end, we'll build a sed script to do each valid prefix in turn.
        #
        for p in ${valid_prefixes}; do
                #
                # When an informational URL was provided, translate it to a
                # hyperlink.  When omitted, simply use the prefix text.
                #
                if [[ -z ${itsinfo["${p}_INFO"]} ]]; then
                        itsinfo["${p}_INFO"]=${p}
                else
                        itsinfo["${p}_INFO"]="<a href=\\\"${itsinfo["${p}_INFO"]
                fi

                #
                # Assume that, for this invocation of webrev, all references
                # to this information tracking system should resolve through
                # the same URL.
                #
                # If the caller specified -O, then always use EXTERNAL_URL.
                #
                # Otherwise, look in the list of domains for a matching
                # INTERNAL_URL.
                #
                [[ -z $Oflag ]] && for d in ${its_domain}; do
                        if [[ -n ${itsinfo["${p}_INTERNAL_URL_${d}"]} ]]; then
                                itsinfo["${p}_URL"]="${itsinfo[${p}_INTERNAL_URL
                                break
                        fi
                done
                if [[ -z ${itsinfo["${p}_URL"]} ]]; then
                        itsinfo["${p}_URL"]="${itsinfo[${p}_EXTERNAL_URL]}"
```

```
                fi

                #
                # Turn the destination URL into a hyperlink
                #
                itsinfo["${p}_URL"]="<a href=\\\"${itsinfo[${p}_URL]}\\\">&</a>"

                # The character class below contains a literal tab
                print "/^${p}[:         ]/ {
                                s;${itsinfo[${p}_REGEX]};${itsinfo[${p}_URL]};g
                                s;^${p};${itsinfo[${p}_INFO]};
                        }" >> ${its_sed_script}
        done

        #
        # The previous loop took care of explicit specification.  Now use
        # the configured priorities to attempt implicit translations.
        #
        for p in ${its_priority}; do
                print "/^${itsinfo[${p}_REGEX]}[        ]/ {
                                s;^${itsinfo[${p}_REGEX]};${itsinfo[${p}_URL]};g
                        }" >> ${its_sed_script}
        done
fi

#
# Search for DO_EVERYTHING above for matching "for" statement
# and explanation of this terminator.
#
done

#
# Output directory.
#
WDIR=${WDIR:-$CWS/webrev}

#
# Name of the webrev, derived from the workspace name or output directory;
# in the future this could potentially be an option.
#
if [[ -n $oflag ]]; then
        WNAME=${WDIR##*/}
else
        WNAME=${CWS##*/}
fi

# Make sure remote target is well formed for remote upload/delete.
if [[ -n $Dflag || -n $Uflag ]]; then
        #
        # If remote target is not specified, build it from scratch using
        # the default values.
        #
        if [[ -z $tflag ]]; then
                remote_target=${DEFAULT_REMOTE_HOST}:${WNAME}
        else
                #
                # Check upload target prefix first.
                #
                if [[ "${remote_target}" != ${rsync_prefix}* &&
                    "${remote_target}" != ${ssh_prefix}* ]]; then
                        print "ERROR: invalid prefix of upload URI" \
                            "($remote_target)"
                        exit 1
                fi
                #
                # If destination specification is not in the form of
                # host_spec:remote_dir then assume it is just remote hostname
                # and append a colon and destination directory formed from
                # local webrev directory name.
                #
                typeset target_no_prefix=${remote_target##*://}
                if [[ ${target_no_prefix} == *:* ]]; then
                        if [[ "${remote_target}" == *: ]]; then
                                remote_target=${remote_target}${WNAME}
                        fi
                else
                        if [[ ${target_no_prefix} == */* ]]; then
                                print "ERROR: badly formed upload URI" \
                                        "($remote_target)"
                                exit 1
                        else
                                remote_target=${remote_target}:${WNAME}
                        fi
                fi
        fi

        #
        # Strip trailing slash. Each upload method will deal with directory
        # specification separately.
        #
        remote_target=${remote_target%/}
fi

#
# Option -D by itself (option -U not present) implies no webrev generation.
#
if [[ -z $Uflag && -n $Dflag ]]; then
        delete_webrev 1 1
        exit $?
fi

#
# Do not generate the webrev, just upload it or delete it.
#
if [[ -n $nflag ]]; then
        if [[ -n $Dflag ]]; then
                delete_webrev 1 1
                (( $? == 0 )) || exit $?
        fi
        if [[ -n $Uflag ]]; then
                upload_webrev
                exit $?
        fi
fi

if [ "${WDIR%%/*}" ]; then
        WDIR=$PWD/$WDIR
fi

if [[ ! -d $WDIR ]]; then
        mkdir -p $WDIR
        (( $? != 0 )) && exit 1
fi

#
# Summarize what we're going to do.
#
print "      Workspace: ${PRETTY_CWS:-$CWS}"
if [[ -n $parent_webrev ]]; then
        print "Compare against: webrev at $parent_webrev"
else
        print "Compare against: ${PRETTY_PWS:-$PWS}"
```

```
fi

[[ -n $INCLUDE_FILE ]] && print "      Including: $INCLUDE_FILE"
print "      Output to: $WDIR"

#
# Save the file list in the webrev dir
#
[[ ! $FLIST -ef $WDIR/file.list ]] && cp $FLIST $WDIR/file.list

rm -f $WDIR/$WNAME.patch
rm -f $WDIR/$WNAME.ps
rm -f $WDIR/$WNAME.pdf

touch $WDIR/$WNAME.patch

print "   Output Files:"

#
# Clean up the file list: Remove comments, blank lines and env variables.
#
$SED -e "s/#.*$//" -e "/=/d" -e "/^[   ]*$/d" $FLIST > /tmp/$$.flist.clean
FLIST=/tmp/$$.flist.clean

#
# For Mercurial, create a cache of manifest entries.
#
if [[ $SCM_MODE == "mercurial" ]]; then
        #
        # Transform the FLIST into a temporary sed script that matches
        # relevant entries in the Mercurial manifest as follows:
        # 1) The script will be used against the parent revision manifest,
        #    so for FLIST lines that have two filenames (a renamed file)
        #    keep only the old name.
        # 2) Escape all forward slashes the filename.
        # 3) Change the filename into another sed command that matches
        #    that file in "hg manifest -v" output:  start of line, three
        #    octal digits for file permissions, space, a file type flag
        #    character, space, the filename, end of line.
        # 4) Eliminate any duplicate entries.  (This can occur if a
        #    file has been used as the source of an hg cp and it's
        #    also been modified in the same changeset.)
        #
        SEDFILE=/tmp/$$.manifest.sed
        $SED '
                s#^[^ ]* ##
                s#/#\\\/#g
                s#^.*$#/^... . &$/p#
        ' < $FLIST | $SORT -u > $SEDFILE

        #
        # Apply the generated script to the output of "hg manifest -v"
        # to get the relevant subset for this webrev.
        #
        HG_PARENT_MANIFEST=/tmp/$$.manifest
        hg -R $CWS manifest -v -r $HG_PARENT |
            $SED -n -f $SEDFILE > $HG_PARENT_MANIFEST
fi

#
# First pass through the files: generate the per-file webrev HTML-files.
#
cat $FLIST | while read LINE
do
        set - $LINE
        P=$1
```

```
	#
	# Normally, each line in the file list is just a pathname of a
	# file that has been modified or created in the child.  A file
	# that is renamed in the child workspace has two names on the
	# line: new name followed by the old name.
	#
	oldname=""
	oldpath=""
	rename=
	if [[ $# -eq 2 ]]; then
		PP=$2			# old filename
		if [[ -f $PP ]]; then
			oldname=" (copied from $PP)"
		else
			oldname=" (renamed from $PP)"
		fi
		oldpath="$PP"
		rename=1
		PDIR=${PP%/*}
		if [[ $PDIR == $PP ]]; then
			PDIR="."   # File at root of workspace
		fi

		PF=${PP##*/}

		DIR=${P%/*}
		if [[ $DIR == $P ]]; then
			DIR="."   # File at root of workspace
		fi

		F=${P##*/}

	else
		DIR=${P%/*}
		if [[ "$DIR" == "$P" ]]; then
			DIR="."   # File at root of workspace
		fi

		F=${P##*/}

		PP=$P
		PDIR=$DIR
		PF=$F
	fi

	COMM=‘getcomments html $P $PP‘

	print "\t$P$oldname\n\t\t\c"

	# Make the webrev mirror directory if necessary
	mkdir -p $WDIR/$DIR

	#
	# We stash old and new files into parallel directories in $WDIR
	# and do our diffs there.  This makes it possible to generate
	# clean looking diffs which don’t have absolute paths present.
	#

	build_old_new "$WDIR" "$PWS" "$PDIR" "$PF" "$CWS" "$DIR" "$F" || \
	    continue

	#
	# Keep the old PWD around, so we can safely switch back after
	# diff generation, such that build_old_new runs in a
	# consistent environment.
```

```
	#
	OWD=$PWD
	cd $WDIR/raw_files
	ofile=old/$PDIR/$PF
	nfile=new/$DIR/$F

	mv_but_nodiff=
	cmp $ofile $nfile > /dev/null 2>&1
	if [[ $? == 0 && $rename == 1 ]]; then
		mv_but_nodiff=1
	fi

	#
	# If we have old and new versions of the file then run the appropriate
	# diffs.  This is complicated by a couple of factors:
	#
	#	- renames must be handled specially: we emit a ’remove’
	#	  diff and an ’add’ diff
	#	- new files and deleted files must be handled specially
	#	- Solaris patch(1m) can’t cope with file creation
	#	  (and hence renames) as of this writing.
	#	- To make matters worse, gnu patch doesn’t interpret the
	#	  output of Solaris diff properly when it comes to
	#	  adds and deletes.  We need to do some "cleansing"
	#	  transformations:
	#	    [to add a file] @@ -1,0 +X,Y @@  -->  @@ -0,0 +X,Y @@
	#	    [to del a file] @@ -X,Y +1,0 @@  -->  @@ -X,Y +0,0 @@
	#
	cleanse_rmfile="$SED ’s/^\(@@ [0-9+,-]*\) [0-9+,-]* @@$/\1 +0,0 @@/’"
	cleanse_newfile="$SED ’s/^@@ [0-9+,-]* \([0-9+,-]* @@\)$/@@ -0,0 \1/’"

	rm -f $WDIR/$DIR/$F.patch
	if [[ -z $rename ]]; then
		if [ ! -f "$ofile" ]; then
			diff -u /dev/null $nfile | sh -c "$cleanse_newfile" \
			    > $WDIR/$DIR/$F.patch
		elif [ ! -f "$nfile" ]; then
			diff -u $ofile /dev/null | sh -c "$cleanse_rmfile" \
			    > $WDIR/$DIR/$F.patch
		else
			diff -u $ofile $nfile > $WDIR/$DIR/$F.patch
		fi
	else
		diff -u $ofile /dev/null | sh -c "$cleanse_rmfile" \
		    > $WDIR/$DIR/$F.patch

		diff -u /dev/null $nfile | sh -c "$cleanse_newfile" \
		    >> $WDIR/$DIR/$F.patch
	fi

	#
	# Tack the patch we just made onto the accumulated patch for the
	# whole wad.
	#
	cat $WDIR/$DIR/$F.patch >> $WDIR/$WNAME.patch

	print " patch\c"

	if [[ -f $ofile && -f $nfile && -z $mv_but_nodiff ]]; then

		${CDIFFCMD:-diff -bt -C 5} $ofile $nfile > $WDIR/$DIR/$F.cdiff
		diff_to_html $F $DIR/$F "C" "$COMM" < $WDIR/$DIR/$F.cdiff \
		    > $WDIR/$DIR/$F.cdiff.html
		print " cdiffs\c"

		${UDIFFCMD:-diff -bt -U 5} $ofile $nfile > $WDIR/$DIR/$F.udiff
```

```
                diff_to_html $F $DIR/$F "U" "$COMM" < $WDIR/$DIR/$F.udiff \
                      > $WDIR/$DIR/$F.udiff.html

                print " udiffs\c"

                if [[ -x $WDIFF ]]; then
                        $WDIFF -c "$COMM" \
                            -t "$WNAME Wdiff $DIR/$F" $ofile $nfile > \
                            $WDIR/$DIR/$F.wdiff.html 2>/dev/null
                        if [[ $? -eq 0 ]]; then
                                print " wdiffs\c"
                        else
                                print " wdiffs[fail]\c"
                        fi
                fi

                sdiff_to_html $ofile $nfile $F $DIR "$COMM" \
                    > $WDIR/$DIR/$F.sdiff.html
                print " sdiffs\c"

                print " frames\c"

                rm -f $WDIR/$DIR/$F.cdiff $WDIR/$DIR/$F.udiff

                difflines $ofile $nfile > $WDIR/$DIR/$F.count

        elif [[ -f $ofile && -f $nfile && -n $mv_but_nodiff ]]; then
                # renamed file: may also have differences
                difflines $ofile $nfile > $WDIR/$DIR/$F.count
        elif [[ -f $nfile ]]; then
                # new file: count added lines
                difflines /dev/null $nfile > $WDIR/$DIR/$F.count
        elif [[ -f $ofile ]]; then
                # old file: count deleted lines
                difflines $ofile /dev/null > $WDIR/$DIR/$F.count
        fi

        #
        # Now we generate the postscript for this file.  We generate diffs
        # only in the event that there is delta, or the file is new (it seems
        # tree-killing to print out the contents of deleted files).
        #
        if [[ -f $nfile ]]; then
                ocr=$ofile
                [[ ! -f $ofile ]] && ocr=/dev/null

                if [[ -z $mv_but_nodiff ]]; then
                        textcomm=‘getcomments text $P $PP‘
                        if [[ -x $CODEREVIEW ]]; then
                                $CODEREVIEW -y "$textcomm" \
                                    -e $ocr $nfile \
                                    > /tmp/$$.psfile 2>/dev/null &&
                                    cat /tmp/$$.psfile >> $WDIR/$WNAME.ps
                                if [[ $? -eq 0 ]]; then
                                        print " ps\c"
                                else
                                        print " ps[fail]\c"
                                fi
                        fi
                fi
        fi

        if [[ -f $ofile ]]; then
                source_to_html Old $PP < $ofile > $WDIR/$DIR/$F-.html
                print " old\c"
        fi
```

```

        if [[ -f $nfile ]]; then
                source_to_html New $P < $nfile > $WDIR/$DIR/$F.html
                print " new\c"
        fi

        cd $OWD

        print
done

frame_nav_js > $WDIR/ancnav.js
frame_navigation > $WDIR/ancnav.html

if [[ ! -f $WDIR/$WNAME.ps ]]; then
        print " Generating PDF: Skipped: no output available"
elif [[ -x $CODEREVIEW && -x $PS2PDF ]]; then
        print " Generating PDF: \c"
        fix_postscript $WDIR/$WNAME.ps | $PS2PDF - > $WDIR/$WNAME.pdf
        print "Done."
else
        print " Generating PDF: Skipped: missing ’ps2pdf’ or ’codereview’"
fi

# If we’re in OpenSolaris mode and there’s a closed dir under $WDIR,
# delete it - prevent accidental publishing of closed source

if [[ -n "$Oflag" ]]; then
        $FIND $WDIR -type d -name closed -exec /bin/rm -rf {} \;
fi

# Now build the index.html file that contains
# links to the source files and their diffs.

cd $CWS

# Save total changed lines for Code Inspection.
print "$TOTL" > $WDIR/TotalChangedLines

print "     index.html: \c"
INDEXFILE=$WDIR/index.html
exec 3<&1                       # duplicate stdout to FD3.
exec 1<&-                       # Close stdout.
exec > $INDEXFILE               # Open stdout to index file.

print "$HTML<head>$STDHEAD"
print "<title>$WNAME</title>"
print "</head>"
print "<body id=\"SUNWwebrev\">"
print "<div class=\"summary\">"
print "<h2>Code Review for $WNAME</h2>"

print "<table>"

#
# Get the preparer’s name:
#
# If the SCM detected is Mercurial, and the configuration property
# ui.username is available, use that, but be careful to properly escape
# angle brackets (HTML syntax characters) in the email address.
#
# Otherwise, use the current userid in the form "John Doe (jdoe)", but
# to maintain compatibility with passwd(4), we must support ’&’ substitutions.
#
preparer=
if [[ "$SCM_MODE" == mercurial ]]; then
```

```
        preparer=‘hg showconfig ui.username 2>/dev/null‘
        if [[ -n "$preparer" ]]; then
                preparer="$(echo "$preparer" | html_quote)"
        fi
fi
if [[ -z "$preparer" ]]; then
        preparer=$(
            $PERL -e '
                ($login, $pw, $uid, $gid, $quota, $cmt, $gcos) = getpwuid($<);
                if ($login) {
                    $gcos =~ s/\&/ucfirst($login)/e;
                    printf "%s (%s)\n", $gcos, $login;
                } else {
                    printf "(unknown)\n";
                }
        ')
fi

PREPDATE=$(LC_ALL=C /usr/bin/date +%Y-%b-%d\ %R\ %z\ %Z)
print "<tr><th>Prepared by:</th><td>$preparer on $PREPDATE</td></tr>"
print "<tr><th>Workspace:</th><td>${PRETTY_CWS:-$CWS}"
print "</td></tr>"
print "<tr><th>Compare against:</th><td>"
if [[ -n $parent_webrev ]]; then
        print "webrev at $parent_webrev"
else
        print "${PRETTY_PWS:-$PWS}"
fi
print "</td></tr>"
print "<tr><th>Summary of changes:</th><td>"
printCI $TOTL $TINS $TDEL $TMOD $TUNC
print "</td></tr>"

if [[ -f $WDIR/$WNAME.patch ]]; then
        wpatch_url="$(print $WNAME.patch | url_encode)"
        print "<tr><th>Patch of changes:</th><td>"
        print "<a href=\"$wpatch_url\">$WNAME.patch</a></td></tr>"
fi
if [[ -f $WDIR/$WNAME.pdf ]]; then
        wpdf_url="$(print $WNAME.pdf | url_encode)"
        print "<tr><th>Printable review:</th><td>"
        print "<a href=\"$wpdf_url\">$WNAME.pdf</a></td></tr>"
fi

if [[ -n "$iflag" ]]; then
        print "<tr><th>Author comments:</th><td><div>"
        cat /tmp/$$.include
        print "</div></td></tr>"
fi
print "</table>"
print "</div>"

#
# Second pass through the files: generate the rest of the index file
#
cat $FLIST | while read LINE
do
        set - $LINE
        P=$1

        if [[ $# == 2 ]]; then
                PP=$2
                oldname="$PP"
        else
                PP=$P
                oldname=""
```

```
        fi

        mv_but_nodiff=
        cmp $WDIR/raw_files/old/$PP $WDIR/raw_files/new/$P > /dev/null 2>&1
        if [[ $? == 0 && -n "$oldname" ]]; then
                mv_but_nodiff=1
        fi

        DIR=${P%/*}
        if [[ $DIR == $P ]]; then
                DIR="."   # File at root of workspace
        fi

        # Avoid processing the same file twice.
        # It’s possible for renamed files to
        # appear twice in the file list

        F=$WDIR/$P

        print "<p>"

        # If there’s a diffs file, make diffs links

        if [[ -f $F.cdiff.html ]]; then
                cdiff_url="$(print $P.cdiff.html | url_encode)"
                udiff_url="$(print $P.udiff.html | url_encode)"
                print "<a href=\"$cdiff_url\">Cdiffs</a>"
                print "<a href=\"$udiff_url\">Udiffs</a>"

                if [[ -f $F.wdiff.html && -x $WDIFF ]]; then
                        wdiff_url="$(print $P.wdiff.html | url_encode)"
                        print "<a href=\"$wdiff_url\">Wdiffs</a>"
                fi

                sdiff_url="$(print $P.sdiff.html | url_encode)"
                print "<a href=\"$sdiff_url\">Sdiffs</a>"

                frames_url="$(print $P.frames.html | url_encode)"
                print "<a href=\"$frames_url\">Frames</a>"
        else
                print " ------ ------ ------"

                if [[ -x $WDIFF ]]; then
                        print " ------"
                fi

                print " ------"
        fi

        # If there’s an old file, make the link

        if [[ -f $F-.html ]]; then
                oldfile_url="$(print $P-.html | url_encode)"
                print "<a href=\"$oldfile_url\">Old</a>"
        else
                print " ---"
        fi

        # If there’s an new file, make the link

        if [[ -f $F.html ]]; then
                newfile_url="$(print $P.html | url_encode)"
                print "<a href=\"$newfile_url\">New</a>"
        else
                print " ---"
        fi
        if [[ -f $F.patch ]]; then
                patch_url="$(print $P.patch | url_encode)"
                print "<a href=\"$patch_url\">Patch</a>"
        else
                print " -----"
        fi

        if [[ -f $WDIR/raw_files/new/$P ]]; then
                rawfiles_url="$(print raw_files/new/$P | url_encode)"
                print "<a href=\"$rawfiles_url\">Raw</a>"
        else
                print " ---"
        fi

        print "<b>$P</b>"

        # For renamed files, clearly state whether or not they are modified
        if [[ -f "$oldname" ]]; then
                if [[ -n "$mv_but_nodiff" ]]; then
                        print "<i>(copied from $oldname)</i>"
                else
                        print "<i>(copied and modified from $oldname)</i>"
                fi
        elif [[ -n "$oldname" ]]; then
                if [[ -n "$mv_but_nodiff" ]]; then
                        print "<i>(renamed from $oldname)</i>"
                else
                        print "<i>(renamed and modified from $oldname)</i>"
                fi
        fi

        # If there’s an old file, but no new file, the file was deleted
        if [[ -f $F-.html && ! -f $F.html ]]; then
                print " <i>(deleted)</i>"
        fi

        #
        # Check for usr/closed and deleted_files/usr/closed
        #
        if [ ! -z "$Oflag" ]; then
                if [[ $P == usr/closed/* || \
                    $P == deleted_files/usr/closed/* ]]; then
                        print "&nbsp;&nbsp;<i>Closed source: omitted from" \
                            "this review</i>"
                fi
        fi

        print "</p>"
        # Insert delta comments

        print "<blockquote><pre>"
        getcomments html $P $PP
        print "</pre>"

        # Add additional comments comment

        print "<!-- Add comments to explain changes in $P here -->"

        # Add count of changes.

        if [[ -f $F.count ]]; then
            cat $F.count
            rm $F.count
        fi
```

```
        if [[ $SCM_MODE == "teamware" ||
            $SCM_MODE == "mercurial" ||
            $SCM_MODE == "unknown" ]]; then

                # Include warnings for important file mode situations:
                # 1) New executable files
                # 2) Permission changes of any kind
                # 3) Existing executable files

                old_mode=
                if [[ -f $WDIR/raw_files/old/$PP ]]; then
                        old_mode=‘get_file_mode $WDIR/raw_files/old/$PP‘
                fi

                new_mode=
                if [[ -f $WDIR/raw_files/new/$P ]]; then
                        new_mode=‘get_file_mode $WDIR/raw_files/new/$P‘
                fi

                if [[ -z "$old_mode" && "$new_mode" = *[1357]* ]]; then
                        print "<span class=\"chmod\">"
                        print "<p>new executable file: mode $new_mode</p>"
                        print "</span>"
                elif [[ -n "$old_mode" && -n "$new_mode" &&
                    "$old_mode" != "$new_mode" ]]; then
                        print "<span class=\"chmod\">"
                        print "<p>mode change: $old_mode to $new_mode</p>"
                        print "</span>"
                elif [[ "$new_mode" = *[1357]* ]]; then
                        print "<span class=\"chmod\">"
                        print "<p>executable file: mode $new_mode</p>"
                        print "</span>"
                fi
        fi

        print "</blockquote>"
done

print
print
print "<hr></hr>"
print "<p style=\"font-size: small\">"
print "This code review page was prepared using <b>$0</b>."
print "Webrev is maintained by the <a href=\"http://www.illumos.org\">"
print "illumos</a> project.  The latest version may be obtained"
print "<a href=\"http://src.illumos.org/source/xref/illumos-gate/usr/src/tools/s
print "</body>"
print "</html>"

exec 1<&-                       # Close FD 1.
exec 1<&3                       # dup FD 3 to restore stdout.
exec 3<&-                       # close FD 3.

print "Done."

#
# If remote deletion was specified and fails do not continue.
#
if [[ -n $Dflag ]]; then
        delete_webrev 1 1
        (( $? == 0 )) || exit $?
fi

if [[ -n $Uflag ]]; then
        upload_webrev
        exit $?
```

```
fi
```