```
**********************************************************
   91558 Thu Dec 12 19:17:00 2013
new/usr/src/tools/scripts/webrev.sh
4389 webrev should fetch preparer info from git if available
**********************************************************
```

```
#!/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 Ivan Richwalski <ivan@seppuku.net>

#
# 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.
```

```
2309 # We don't warn on the absence of 'wx' until later when we've
2310 # determined that we actually need to try to invoke it.
2311 #
2312 [[ ! -x $CODEREVIEW ]] && print -u2 "WARNING: codereview(1) not found."
2313 [[ ! -x $PS2PDF ]] && print -u2 "WARNING: ps2pdf(1) not found."
2314 [[ ! -x $WDIFF ]] && print -u2 "WARNING: wdiff not found."

2316 # Declare global total counters.
2317 integer TOTL TINS TDEL TMOD TUNC

2319 # default remote host for upload/delete
2320 typeset -r DEFAULT_REMOTE_HOST="cr.opensolaris.org"
2321 # prefixes for upload targets
2322 typeset -r rsync_prefix="rsync://"
2323 typeset -r ssh_prefix="ssh://"

2325 Cflag=
2326 Dflag=
2327 flist_mode=
2328 flist_file=
2329 iflag=
2330 Iflag=
2331 lflag=
2332 Nflag=
2333 nflag=
2334 Oflag=
2335 oflag=
2336 pflag=
2337 tflag=
2338 uflag=
2339 Uflag=
2340 wflag=
2341 remote_target=

2343 #
2344 # NOTE: when adding/removing options it is necessary to sync the list
2345 #       with usr/src/tools/onbld/hgext/cdm.py
2346 #
2347 while getopts "C:Di:I:lnNo:Op:t:Uw" opt
2348 do
2349         case $opt in
2350         C)      Cflag=1
2351                 ITSCONF=$OPTARG;;

2353         D)      Dflag=1;;

2355         i)      iflag=1
2356                 INCLUDE_FILE=$OPTARG;;

2358         I)      Iflag=1
2359                 ITSREG=$OPTARG;;

2361         #
2362         # If -l has been specified, we need to abort further options
2363         # processing, because subsequent arguments are going to be
2364         # arguments to 'putback -n'.
2365         #
2366         l)      lflag=1
2367                 break;;

2369         N)      Nflag=1;;

2371         n)      nflag=1;;

2373         O)      Oflag=1;;
```



```
2375         o)      oflag=1
2376                 # Strip the trailing slash to correctly form remote target.
2377                 WDIR=${OPTARG%/};;

2379         p)      pflag=1
2380                 codemgr_parent=$OPTARG;;

2382         t)      tflag=1
2383                 remote_target=$OPTARG;;

2385         U)      Uflag=1;;

2387         w)      wflag=1;;

2389         ?)      usage;;
2390         esac
2391 done

2393 FLIST=/tmp/$$.flist

2395 if [[ -n $wflag && -n $lflag ]]; then
2396         usage
2397 fi

2399 # more sanity checking
2400 if [[ -n $nflag && -z $Uflag ]]; then
2401         print "it does not make sense to skip webrev generation" \
2402             "without -U"
2403         exit 1
2404 fi

2406 if [[ -n $tflag && -z $Uflag && -z $Dflag ]]; then
2407         echo "remote target has to be used only for upload or delete"
2408         exit 1
2409 fi

2411 #
2412 # For the invocation "webrev -n -U" with no other options, webrev will assume
2413 # that the webrev exists in ${CWS}/webrev, but will upload it using the name
2414 # $(basename ${CWS}).  So we need to get CWS set before we skip any remaining
2415 # logic.
2416 #
2417 $WHICH_SCM | read SCM_MODE junk || exit 1
2418 if [[ $SCM_MODE == "teamware" ]]; then
2419         #
2420         # Teamware priorities:
2421         # 1. CODEMGR_WS from the environment
2422         # 2. workspace name
2423         #
2424         [[ -z $codemgr_ws && -n $CODEMGR_WS ]] && codemgr_ws=$CODEMGR_WS
2425         if [[ -n $codemgr_ws && ! -d $codemgr_ws ]]; then
2426                 print -u2 "$codemgr_ws: no such workspace"
2427                 exit 1
2428         fi
2429         [[ -z $codemgr_ws ]] && codemgr_ws=$(workspace name)
2430         codemgr_ws=$(cd $codemgr_ws;print $PWD)
2431         CODEMGR_WS=$codemgr_ws
2432         CWS=$codemgr_ws
2433 elif [[ $SCM_MODE == "mercurial" ]]; then
2434         #
2435         # Mercurial priorities:
2436         # 1. hg root from CODEMGR_WS environment variable
2437         # 1a. hg root from CODEMGR_WS/usr/closed if we're somewhere under
2438         #    usr/closed when we run webrev
2439         # 2. hg root from directory of invocation
2440         #
```

```
2441         if [[ ${PWD} =~ "usr/closed" ]]; then
2442                 testparent=${CODEMGR_WS}/usr/closed
2443                 # If we're in OpenSolaris mode, we enforce a minor policy:
2444                 # help to make sure the reviewer doesn't accidentally publish
2445                 # source which is under usr/closed
2446                 if [[ -n "$Oflag" ]]; then
2447                         print -u2 "OpenSolaris output not permitted with" \
2448                             "usr/closed changes"
2449                         exit 1
2450                 fi
2451         else
2452                 testparent=${CODEMGR_WS}
2453         fi
2454         [[ -z $codemgr_ws && -n $testparent ]] && \
2455             codemgr_ws=$(hg root -R $testparent 2>/dev/null)
2456         [[ -z $codemgr_ws ]] && codemgr_ws=$(hg root 2>/dev/null)
2457         CWS=$codemgr_ws
2458 elif [[ $SCM_MODE == "git" ]]; then
2459         #
2460         # Git priorities:
2461         # 1. git rev-parse --git-dir from CODEMGR_WS environment variable
2462         # 2. git rev-parse --git-dir from directory of invocation
2463         #
2464         [[ -z $codemgr_ws && -n $CODEMGR_WS ]] && \
2465             codemgr_ws=$($GIT --git-dir=$CODEMGR_WS/.git rev-parse --git-dir \
2466                 2>/dev/null)
2467         [[ -z $codemgr_ws ]] && \
2468             codemgr_ws=$($GIT rev-parse --git-dir 2>/dev/null)

2470         if [[ "$codemgr_ws" == ".git" ]]; then
2471                 codemgr_ws="${PWD}/${codemgr_ws}"
2472         fi

2474         codemgr_ws=$(dirname $codemgr_ws) # Lose the '/.git'
2475         CWS="$codemgr_ws"
2476 elif [[ $SCM_MODE == "subversion" ]]; then
2477         #
2478         # Subversion priorities:
2479         # 1. CODEMGR_WS from environment
2480         # 2. Relative path from current directory to SVN repository root
2481         #
2482         if [[ -n $CODEMGR_WS && -d $CODEMGR_WS/.svn ]]; then
2483                 CWS=$CODEMGR_WS
2484         else
2485                 svn info | while read line; do
2486                         if [[ $line == "URL: "* ]]; then
2487                                 url=${line#URL: }
2488                         elif [[ $line == "Repository Root: "* ]]; then
2489                                 repo=${line#Repository Root: }
2490                         fi
2491                 done

2493                 rel=${url#$repo}
2494                 CWS=${PWD%$rel}
2495         fi
2496 fi

2498 #
2499 # If no SCM has been determined, take either the environment setting
2500 # setting for CODEMGR_WS, or the current directory if that wasn't set.
2501 #
2502 if [[ -z ${CWS} ]]; then
2503         CWS=${CODEMGR_WS:-.}
2504 fi

2506 #
```



```
2507 # If the command line options indicate no webrev generation, either
2508 # explicitly (-n) or implicitly (-D but not -U), then there's a whole
2509 # ton of logic we can skip.
2510 #
2511 # Instead of increasing indentation, we intentionally leave this loop
2512 # body open here, and exit via break from multiple points within.
2513 # Search for DO_EVERYTHING below to find the break points and closure.
2514 #
2515 for do_everything in 1; do

2517 # DO_EVERYTHING: break point
2518 if [[ -n $nflag || ( -z $Uflag && -n $Dflag ) ]]; then
2519         break
2520 fi

2522 #
2523 # If this manually set as the parent, and it appears to be an earlier webrev,
2524 # then note that fact and set the parent to the raw_files/new subdirectory.
2525 #
2526 if [[ -n $pflag && -d $codemgr_parent/raw_files/new ]]; then
2527         parent_webrev=$(readlink -f "$codemgr_parent")
2528         codemgr_parent=$(readlink -f "$codemgr_parent/raw_files/new")
2529 fi

2531 if [[ -z $wflag && -z $lflag ]]; then
2532         shift $(($OPTIND - 1))

2534         if [[ $1 == "-" ]]; then
2535                 cat > $FLIST
2536                 flist_mode="stdin"
2537                 flist_done=1
2538                 shift
2539         elif [[ -n $1 ]]; then
2540                 if [[ ! -r $1 ]]; then
2541                         print -u2 "$1: no such file or not readable"
2542                         usage
2543                 fi
2544                 cat $1 > $FLIST
2545                 flist_mode="file"
2546                 flist_file=$1
2547                 flist_done=1
2548                 shift
2549         else
2550                 flist_mode="auto"
2551         fi
2552 fi

2554 #
2555 # Before we go on to further consider -l and -w, work out which SCM we think
2556 # is in use.
2557 #
2558 case "$SCM_MODE" in
2559 teamware|mercurial|git|subversion)
2560         ;;
2561 unknown)
2562         if [[ $flist_mode == "auto" ]]; then
2563                 print -u2 "Unable to determine SCM in use and file list not spec
2564                 print -u2 "See which_scm(1) for SCM detection information."
2565                 exit 1
2566         fi
2567         ;;
2568 *)
2569         if [[ $flist_mode == "auto" ]]; then
2570                 print -u2 "Unsupported SCM in use ($SCM_MODE) and file list not
2571                 exit 1
2572         fi
```

```
2573         ;;
2574 esac

2576 print -u2 "   SCM detected: $SCM_MODE"

2578 if [[ -n $lflag ]]; then
2579         #
2580         # If the -l flag is given instead of the name of a file list,
2581         # then generate the file list by extracting file names from a
2582         # putback -n.
2583         #
2584         shift $(($OPTIND - 1))
2585         if [[ $SCM_MODE == "teamware" ]]; then
2586                 flist_from_teamware "$*"
2587         else
2588                 print -u2 -- "Error: -l option only applies to TeamWare"
2589                 exit 1
2590         fi
2591         flist_done=1
2592         shift $#
2593 elif [[ -n $wflag ]]; then
2594         #
2595         # If the -w is given then assume the file list is in Bonwick's "wx"
2596         # command format, i.e.  pathname lines alternating with SCCS comment
2597         # lines with blank lines as separators.  Use the SCCS comments later
2598         # in building the index.html file.
2599         #
2600         shift $((OPTIND - 1))
2601         wxfile=$1
2602         if [[ -z $wxfile && -n $CODEMGR_WS ]]; then
2603                 if [[ -r $CODEMGR_WS/wx/active ]]; then
2604                         wxfile=$CODEMGR_WS/wx/active
2605                 fi
2606         fi

2608         [[ -z $wxfile ]] && print -u2 "wx file not specified, and could not " \
2609             "be auto-detected (check \$CODEMGR_WS)" && exit 1

2611         if [[ ! -r $wxfile ]]; then
2612                 print -u2 "$wxfile: no such file or not readable"
2613                 usage
2614         fi

2616         print -u2 " File list from: wx 'active' file '$wxfile' ... \c"
2617         flist_from_wx $wxfile
2618         flist_done=1
2619         if [[ -n "$*" ]]; then
2620                 shift
2621         fi
2622 elif [[ $flist_mode == "stdin" ]]; then
2623         print -u2 " File list from: standard input"
2624 elif [[ $flist_mode == "file" ]]; then
2625         print -u2 " File list from: $flist_file"
2626 fi

2628 if [[ $# -gt 0 ]]; then
2629         print -u2 "WARNING: unused arguments: $*"
2630 fi

2632 #
2633 # Before we entered the DO_EVERYTHING loop, we should have already set CWS
2634 # and CODEMGR_WS as needed.  Here, we set the parent workspace.
2635 #

2637 if [[ $SCM_MODE == "teamware" ]]; then
```



```
2639         #
2640         # Teamware priorities:
2641         #
2642         #      1) via -p command line option
2643         #      2) in the user environment
2644         #      3) in the flist
2645         #      4) automatically based on the workspace
2646         #

2648         #
2649         # For 1, codemgr_parent will already be set.  Here's 2:
2650         #
2651         [[ -z $codemgr_parent && -n $CODEMGR_PARENT ]] && \
2652             codemgr_parent=$CODEMGR_PARENT
2653         if [[ -n $codemgr_parent && ! -d $codemgr_parent ]]; then
2654                 print -u2 "$codemgr_parent: no such directory"
2655                 exit 1
2656         fi

2658         #
2659         # If we're in auto-detect mode and we haven't already gotten the file
2660         # list, then see if we can get it by probing for wx.
2661         #
2662         if [[ -z $flist_done && $flist_mode == "auto" && -n $codemgr_ws ]]; then
2663                 if [[ ! -x $WX ]]; then
2664                         print -u2 "WARNING: wx not found!"
2665                 fi

2667                 #
2668                 # We need to use wx list -w so that we get renamed files, etc.
2669                 # but only if a wx active file exists-- otherwise wx will
2670                 # hang asking us to initialize our wx information.
2671                 #
2672                 if [[ -x $WX && -f $codemgr_ws/wx/active ]]; then
2673                         print -u2 " File list from: 'wx list -w' ... \c"
2674                         $WX list -w > $FLIST
2675                         $WX comments > /tmp/$$.wx_comments
2676                         wxfile=/tmp/$$.wx_comments
2677                         print -u2 "done"
2678                         flist_done=1
2679                 fi
2680         fi

2682         #
2683         # If by hook or by crook we've gotten a file list by now (perhaps
2684         # from the command line), eval it to extract environment variables from
2685         # it: This is method 3 for finding the parent.
2686         #
2687         if [[ -z $flist_done ]]; then
2688                 flist_from_teamware
2689         fi
2690         env_from_flist

2692         #
2693         # (4) If we still don't have a value for codemgr_parent, get it
2694         # from workspace.
2695         #
2696         [[ -z $codemgr_parent ]] && codemgr_parent=`workspace parent`
2697         if [[ ! -d $codemgr_parent ]]; then
2698                 print -u2 "$CODEMGR_PARENT: no such parent workspace"
2699                 exit 1
2700         fi

2702         PWS=$codemgr_parent

2704         [[ -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.
```

```
2837         #
2838         if [[ -n $flist_done ]]; then
2839                 env_from_flist
2840         fi

2842         #
2843         # If we don't have a wx-format file list, build one we can pull change
2844         # comments from.
2845         #
2846         if [[ -z $wxfile ]]; then
2847                 print "  Comments from: git...\c"
2848                 git_wxfile "$CWS" "$real_parent"
2849                 print " Done."
2850         fi

2852         if [[ -z $GIT_PARENT ]]; then
2853                 GIT_PARENT=$($GIT merge-base "$real_parent" HEAD)
2854         fi
2855         if [[ -z $GIT_PARENT ]]; then
2856                 print -u2 "Error: Cannot discover parent revision"
2857                 exit 1
2858         fi

2860         pnode=$(trim_digest $GIT_PARENT)

2862         if [[ $real_parent == */* ]]; then
2863                 origin=$(echo $real_parent | cut -d/ -f1)
2864                 origin=$($GIT remote -v | \
2865                     $AWK '$1 == "'$origin'" { print $2; exit }')
2866                 PRETTY_PWS="${PWS} (${origin} at ${pnode})"
2867         else
2868                 PRETTY_PWS="${PWS} (at ${pnode})"
2869         fi

2871         cnode=$($GIT --git-dir=${codemgr_ws}/.git rev-parse --short=12 HEAD \
2872             2>/dev/null)
2873         PRETTY_CWS="${CWS} (at ${cnode})"
2874 elif [[ $SCM_MODE == "subversion" ]]; then

2876         #
2877         # We only will have a real parent workspace in the case one
2878         # was specified (be it an older webrev, or another checkout).
2879         #
2880         [[ -n $codemgr_parent ]] && PWS=$codemgr_parent

2882         if [[ -z $flist_done && $flist_mode == "auto" ]]; then
2883                 flist_from_subversion $CWS $OLDPWD
2884         fi
2885 else
2886     if [[ $SCM_MODE == "unknown" ]]; then
2887         print -u2 "    Unknown type of SCM in use"
2888     else
2889         print -u2 "    Unsupported SCM in use: $SCM_MODE"
2890     fi

2892     env_from_flist

2894     if [[ -z $CODEMGR_WS ]]; then
2895         print -u2 "SCM not detected/supported and CODEMGR_WS not specified"
2896         exit 1
2897     fi

2899     if [[ -z $CODEMGR_PARENT ]]; then
2900         print -u2 "SCM not detected/supported and CODEMGR_PARENT not specified"
2901         exit 1
2902     fi
```



```
2904     CWS=$CODEMGR_WS
2905     PWS=$CODEMGR_PARENT
2906 fi

2908 #
2909 # If the user didn't specify a -i option, check to see if there is a
2910 # webrev-info file in the workspace directory.
2911 #
2912 if [[ -z $iflag && -r "$CWS/webrev-info" ]]; then
2913         iflag=1
2914         INCLUDE_FILE="$CWS/webrev-info"
2915 fi

2917 if [[ -n $iflag ]]; then
2918         if [[ ! -r $INCLUDE_FILE ]]; then
2919                 print -u2 "include file '$INCLUDE_FILE' does not exist or is" \
2920                     "not readable."
2921                 exit 1
2922         else
2923                 #
2924                 # $INCLUDE_FILE may be a relative path, and the script alters
2925                 # PWD, so we just stash a copy in /tmp.
2926                 #
2927                 cp $INCLUDE_FILE /tmp/$$.include
2928         fi
2929 fi

2931 # DO_EVERYTHING: break point
2932 if [[ -n $Nflag ]]; then
2933         break
2934 fi

2936 typeset -A itsinfo
2937 typeset -r its_sed_script=/tmp/$$.its_sed
2938 valid_prefixes=
2939 if [[ -z $nflag ]]; then
2940         DEFREGFILE="$(/bin/dirname "$(whence $0)")/../etc/its.reg"
2941         if [[ -n $Iflag ]]; then
2942                 REGFILE=$ITSREG
2943         elif [[ -r $HOME/.its.reg ]]; then
2944                 REGFILE=$HOME/.its.reg
2945         else
2946                 REGFILE=$DEFREGFILE
2947         fi
2948         if [[ ! -r $REGFILE ]]; then
2949                 print "ERROR: Unable to read database registry file $REGFILE"
2950                 exit 1
2951         elif [[ $REGFILE != $DEFREGFILE ]]; then
2952                 print "   its.reg from: $REGFILE"
2953         fi

2955         $SED -e '/^#/d' -e '/^[         ]*$/d' $REGFILE | while read LINE; do

2957                 name=${LINE%%=*}
2958                 value="${LINE#*=}"

2960                 if [[ $name == PREFIX ]]; then
2961                         p=${value}
2962                         valid_prefixes="${p} ${valid_prefixes}"
2963                 else
2964                         itsinfo["${p}_${name}"]="${value}"
2965                 fi
2966         done
```

```
2969         DEFCONFFILE="$(/bin/dirname "$(whence $0)")/../etc/its.conf"
2970         CONFFILES=$DEFCONFFILE
2971         if [[ -r $HOME/.its.conf ]]; then
2972                 CONFFILES="${CONFFILES} $HOME/.its.conf"
2973         fi
2974         if [[ -n $Cflag ]]; then
2975                 CONFFILES="${CONFFILES} ${ITSCONF}"
2976         fi
2977         its_domain=
2978         its_priority=
2979         for cf in ${CONFFILES}; do
2980                 if [[ ! -r $cf ]]; then
2981                         print "ERROR: Unable to read database configuration file
2982                         exit 1
2983                 elif [[ $cf != $DEFCONFFILE ]]; then
2984                         print "       its.conf: reading $cf"
2985                 fi
2986                 $SED -e '/^#/d' -e '/^[         ]*$/d' $cf | while read LINE; do
2987                     eval "${LINE}"
2988                 done
2989         done

2991         #
2992         # If an information tracking system is explicitly identified by prefix,
2993         # we want to disregard the specified priorities and resolve it according
2994         #
2995         # To that end, we'll build a sed script to do each valid prefix in turn.
2996         #
2997         for p in ${valid_prefixes}; do
2998                 #
2999                 # When an informational URL was provided, translate it to a
3000                 # hyperlink.  When omitted, simply use the prefix text.
3001                 #
3002                 if [[ -z ${itsinfo["${p}_INFO"]} ]]; then
3003                         itsinfo["${p}_INFO"]=${p}
3004                 else
3005                         itsinfo["${p}_INFO"]="<a href=\\\"${itsinfo["${p}_INFO"]
3006                 fi

3008                 #
3009                 # Assume that, for this invocation of webrev, all references
3010                 # to this information tracking system should resolve through
3011                 # the same URL.
3012                 #
3013                 # If the caller specified -O, then always use EXTERNAL_URL.
3014                 #
3015                 # Otherwise, look in the list of domains for a matching
3016                 # INTERNAL_URL.
3017                 #
3018                 [[ -z $Oflag ]] && for d in ${its_domain}; do
3019                         if [[ -n ${itsinfo["${p}_INTERNAL_URL_${d}"]} ]]; then
3020                                 itsinfo["${p}_URL"]="${itsinfo[${p}_INTERNAL_URL
3021                                 break
3022                         fi
3023                 done
3024                 if [[ -z ${itsinfo["${p}_URL"]} ]]; then
3025                         itsinfo["${p}_URL"]="${itsinfo[${p}_EXTERNAL_URL]}"
3026                 fi

3028                 #
3029                 # Turn the destination URL into a hyperlink
3030                 #
3031                 itsinfo["${p}_URL"]="<a href=\\\"${itsinfo[${p}_URL]}\\\">&</a>"

3033                 # The character class below contains a literal tab
3034                 print "/^${p}[:         ]/ {
```



```
3035                                 s;${itsinfo[${p}_REGEX]};${itsinfo[${p}_URL]};g
3036                                 s;^${p};${itsinfo[${p}_INFO]};
3037                         }" >> ${its_sed_script}
3038         done

3040         #
3041         # The previous loop took care of explicit specification.  Now use
3042         # the configured priorities to attempt implicit translations.
3043         #
3044         for p in ${its_priority}; do
3045                 print "/^${itsinfo[${p}_REGEX]}[        ]/ {
3046                                 s;^${itsinfo[${p}_REGEX]};${itsinfo[${p}_URL]};g
3047                         }" >> ${its_sed_script}
3048         done
3049 fi

3051 #
3052 # Search for DO_EVERYTHING above for matching "for" statement
3053 # and explanation of this terminator.
3054 #
3055 done

3057 #
3058 # Output directory.
3059 #
3060 WDIR=${WDIR:-$CWS/webrev}

3062 #
3063 # Name of the webrev, derived from the workspace name or output directory;
3064 # in the future this could potentially be an option.
3065 #
3066 if [[ -n $oflag ]]; then
3067         WNAME=${WDIR##*/}
3068 else
3069         WNAME=${CWS##*/}
3070 fi

3072 # Make sure remote target is well formed for remote upload/delete.
3073 if [[ -n $Dflag || -n $Uflag ]]; then
3074         #
3075         # If remote target is not specified, build it from scratch using
3076         # the default values.
3077         #
3078         if [[ -z $tflag ]]; then
3079                 remote_target=${DEFAULT_REMOTE_HOST}:${WNAME}
3080         else
3081                 #
3082                 # Check upload target prefix first.
3083                 #
3084                 if [[ "${remote_target}" != ${rsync_prefix}* &&
3085                     "${remote_target}" != ${ssh_prefix}* ]]; then
3086                         print "ERROR: invalid prefix of upload URI" \
3087                             "($remote_target)"
3088                         exit 1
3089                 fi
3090                 #
3091                 # If destination specification is not in the form of
3092                 # host_spec:remote_dir then assume it is just remote hostname
3093                 # and append a colon and destination directory formed from
3094                 # local webrev directory name.
3095                 #
3096                 typeset target_no_prefix=${remote_target##*://}
3097                 if [[ ${target_no_prefix} == *:* ]]; then
3098                         if [[ "${remote_target}" == *: ]]; then
3099                                 remote_target=${remote_target}${WNAME}
3100                         fi
```

```
3101                 else
3102                         if [[ ${target_no_prefix} == */* ]]; then
3103                                 print "ERROR: badly formed upload URI" \
3104                                     "($remote_target)"
3105                                 exit 1
3106                         else
3107                                 remote_target=${remote_target}:${WNAME}
3108                         fi
3109                 fi
3110         fi

3112         #
3113         # Strip trailing slash. Each upload method will deal with directory
3114         # specification separately.
3115         #
3116         remote_target=${remote_target%/}
3117 fi

3119 #
3120 # Option -D by itself (option -U not present) implies no webrev generation.
3121 #
3122 if [[ -z $Uflag && -n $Dflag ]]; then
3123         delete_webrev 1 1
3124         exit $?
3125 fi

3127 #
3128 # Do not generate the webrev, just upload it or delete it.
3129 #
3130 if [[ -n $nflag ]]; then
3131         if [[ -n $Dflag ]]; then
3132                 delete_webrev 1 1
3133                 (( $? == 0 )) || exit $?
3134         fi
3135         if [[ -n $Uflag ]]; then
3136                 upload_webrev
3137                 exit $?
3138         fi
3139 fi

3141 if [ "${WDIR%%/*}" ]; then
3142         WDIR=$PWD/$WDIR
3143 fi

3145 if [[ ! -d $WDIR ]]; then
3146         mkdir -p $WDIR
3147         (( $? != 0 )) && exit 1
3148 fi

3150 #
3151 # Summarize what we're going to do.
3152 #
3153 print "      Workspace: ${PRETTY_CWS:-$CWS}"
3154 if [[ -n $parent_webrev ]]; then
3155         print "Compare against: webrev at $parent_webrev"
3156 else
3157         print "Compare against: ${PRETTY_PWS:-$PWS}"
3158 fi

3160 [[ -n $INCLUDE_FILE ]] && print "      Including: $INCLUDE_FILE"
3161 print "      Output to: $WDIR"

3163 #
3164 # Save the file list in the webrev dir
3165 #
3166 [[ ! $FLIST -ef $WDIR/file.list ]] && cp $FLIST $WDIR/file.list
```



```
3168 rm -f $WDIR/$WNAME.patch
3169 rm -f $WDIR/$WNAME.ps
3170 rm -f $WDIR/$WNAME.pdf

3172 touch $WDIR/$WNAME.patch

3174 print "   Output Files:"

3176 #
3177 # Clean up the file list: Remove comments, blank lines and env variables.
3178 #
3179 $SED -e "s/#.*$//" -e "/=/d" -e "/^[   ]*$/d" $FLIST > /tmp/$$.flist.clean
3180 FLIST=/tmp/$$.flist.clean

3182 #
3183 # For Mercurial, create a cache of manifest entries.
3184 #
3185 if [[ $SCM_MODE == "mercurial" ]]; then
3186         #
3187         # Transform the FLIST into a temporary sed script that matches
3188         # relevant entries in the Mercurial manifest as follows:
3189         # 1) The script will be used against the parent revision manifest,
3190         #    so for FLIST lines that have two filenames (a renamed file)
3191         #    keep only the old name.
3192         # 2) Escape all forward slashes the filename.
3193         # 3) Change the filename into another sed command that matches
3194         #    that file in "hg manifest -v" output:  start of line, three
3195         #    octal digits for file permissions, space, a file type flag
3196         #    character, space, the filename, end of line.
3197         # 4) Eliminate any duplicate entries.  (This can occur if a
3198         #    file has been used as the source of an hg cp and it's
3199         #    also been modified in the same changeset.)
3200         #
3201         SEDFILE=/tmp/$$.manifest.sed
3202         $SED '
3203                 s#^[^ ]* ##
3204                 s#/#\\\/#g
3205                 s#^.*$#/^... . &$/p#
3206         ' < $FLIST | $SORT -u > $SEDFILE

3208         #
3209         # Apply the generated script to the output of "hg manifest -v"
3210         # to get the relevant subset for this webrev.
3211         #
3212         HG_PARENT_MANIFEST=/tmp/$$.manifest
3213         hg -R $CWS manifest -v -r $HG_PARENT |
3214             $SED -n -f $SEDFILE > $HG_PARENT_MANIFEST
3215 fi

3217 #
3218 # First pass through the files: generate the per-file webrev HTML-files.
3219 #
3220 cat $FLIST | while read LINE
3221 do
3222         set - $LINE
3223         P=$1

3225         #
3226         # Normally, each line in the file list is just a pathname of a
3227         # file that has been modified or created in the child.  A file
3228         # that is renamed in the child workspace has two names on the
3229         # line: new name followed by the old name.
3230         #
3231         oldname=""
3232         oldpath=""
```

```
3233         rename=
3234         if [[ $# -eq 2 ]]; then
3235                 PP=$2                   # old filename
3236                 if [[ -f $PP ]]; then
3237                         oldname=" (copied from $PP)"
3238                 else
3239                         oldname=" (renamed from $PP)"
3240                 fi
3241                 oldpath="$PP"
3242                 rename=1
3243                 PDIR=${PP%/*}
3244                 if [[ $PDIR == $PP ]]; then
3245                         PDIR="."   # File at root of workspace
3246                 fi

3248                 PF=${PP##*/}

3250                 DIR=${P%/*}
3251                 if [[ $DIR == $P ]]; then
3252                         DIR="."   # File at root of workspace
3253                 fi

3255                 F=${P##*/}

3257         else
3258                 DIR=${P%/*}
3259                 if [[ "$DIR" == "$P" ]]; then
3260                         DIR="."   # File at root of workspace
3261                 fi

3263                 F=${P##*/}

3265                 PP=$P
3266                 PDIR=$DIR
3267                 PF=$F
3268         fi

3270         COMM=‘getcomments html $P $PP‘

3272         print "\t$P$oldname\n\t\t\c"

3274         # Make the webrev mirror directory if necessary
3275         mkdir -p $WDIR/$DIR

3277         #
3278         # We stash old and new files into parallel directories in $WDIR
3279         # and do our diffs there.  This makes it possible to generate
3280         # clean looking diffs which don’t have absolute paths present.
3281         #

3283         build_old_new "$WDIR" "$PWS" "$PDIR" "$PF" "$CWS" "$DIR" "$F" || \
3284             continue

3286         #
3287         # Keep the old PWD around, so we can safely switch back after
3288         # diff generation, such that build_old_new runs in a
3289         # consistent environment.
3290         #
3291         OWD=$PWD
3292         cd $WDIR/raw_files
3293         ofile=old/$PDIR/$PF
3294         nfile=new/$DIR/$F

3296         mv_but_nodiff=
3297         cmp $ofile $nfile > /dev/null 2>&1
3298         if [[ $? == 0 && $rename == 1 ]]; then
```



```
3299                 mv_but_nodiff=1
3300         fi

3302         #
3303         # If we have old and new versions of the file then run the appropriate
3304         # diffs.  This is complicated by a couple of factors:
3305         #
3306         #       - renames must be handled specially: we emit a ’remove’
3307         #         diff and an ’add’ diff
3308         #       - new files and deleted files must be handled specially
3309         #       - Solaris patch(1m) can’t cope with file creation
3310         #         (and hence renames) as of this writing.
3311         #       - To make matters worse, gnu patch doesn’t interpret the
3312         #         output of Solaris diff properly when it comes to
3313         #         adds and deletes.  We need to do some "cleansing"
3314         #         transformations:
3315         #           [to add a file] @@ -1,0 +X,Y @@  -->  @@ -0,0 +X,Y @@
3316         #           [to del a file] @@ -X,Y +1,0 @@  -->  @@ -X,Y +0,0 @@
3317         #
3318         cleanse_rmfile="$SED ’s/^\(@@ [0-9+,-]*\) [0-9+,-]* @@$/\1 +0,0 @@/’"
3319         cleanse_newfile="$SED ’s/^@@ [0-9+,-]* \([0-9+,-]* @@\)$/@@ -0,0 \1/’"

3321         rm -f $WDIR/$DIR/$F.patch
3322         if [[ -z $rename ]]; then
3323                 if [ ! -f "$ofile" ]; then
3324                         diff -u /dev/null $nfile | sh -c "$cleanse_newfile" \
3325                             > $WDIR/$DIR/$F.patch
3326                 elif [ ! -f "$nfile" ]; then
3327                         diff -u $ofile /dev/null | sh -c "$cleanse_rmfile" \
3328                             > $WDIR/$DIR/$F.patch
3329                 else
3330                         diff -u $ofile $nfile > $WDIR/$DIR/$F.patch
3331                 fi
3332         else
3333                 diff -u $ofile /dev/null | sh -c "$cleanse_rmfile" \
3334                     > $WDIR/$DIR/$F.patch

3336                 diff -u /dev/null $nfile | sh -c "$cleanse_newfile" \
3337                     >> $WDIR/$DIR/$F.patch
3338         fi

3340         #
3341         # Tack the patch we just made onto the accumulated patch for the
3342         # whole wad.
3343         #
3344         cat $WDIR/$DIR/$F.patch >> $WDIR/$WNAME.patch

3346         print " patch\c"

3348         if [[ -f $ofile && -f $nfile && -z $mv_but_nodiff ]]; then

3350                 ${CDIFFCMD:-diff -bt -C 5} $ofile $nfile > $WDIR/$DIR/$F.cdiff
3351                 diff_to_html $F $DIR/$F "C" "$COMM" < $WDIR/$DIR/$F.cdiff \
3352                     > $WDIR/$DIR/$F.cdiff.html
3353                 print " cdiffs\c"

3355                 ${UDIFFCMD:-diff -bt -U 5} $ofile $nfile > $WDIR/$DIR/$F.udiff
3356                 diff_to_html $F $DIR/$F "U" "$COMM" < $WDIR/$DIR/$F.udiff \
3357                     > $WDIR/$DIR/$F.udiff.html

3359                 print " udiffs\c"

3361                 if [[ -x $WDIFF ]]; then
3362                         $WDIFF -c "$COMM" \
3363                             -t "$WNAME Wdiff $DIR/$F" $ofile $nfile > \
3364                             $WDIR/$DIR/$F.wdiff.html 2>/dev/null
```

```
3365                        if [[ $? -eq 0 ]]; then
3366                                print " wdiffs\c"
3367                        else
3368                                print " wdiffs[fail]\c"
3369                        fi
3370                fi

3372                sdiff_to_html $ofile $nfile $F $DIR "$COMM" \
3373                    > $WDIR/$DIR/$F.sdiff.html
3374                print " sdiffs\c"

3376                print " frames\c"

3378                rm -f $WDIR/$DIR/$F.cdiff $WDIR/$DIR/$F.udiff

3380                difflines $ofile $nfile > $WDIR/$DIR/$F.count

3382        elif [[ -f $ofile && -f $nfile && -n $mv_but_nodiff ]]; then
3383                # renamed file: may also have differences
3384                difflines $ofile $nfile > $WDIR/$DIR/$F.count
3385        elif [[ -f $nfile ]]; then
3386                # new file: count added lines
3387                difflines /dev/null $nfile > $WDIR/$DIR/$F.count
3388        elif [[ -f $ofile ]]; then
3389                # old file: count deleted lines
3390                difflines $ofile /dev/null > $WDIR/$DIR/$F.count
3391        fi

3393        #
3394        # Now we generate the postscript for this file.  We generate diffs
3395        # only in the event that there is delta, or the file is new (it seems
3396        # tree-killing to print out the contents of deleted files).
3397        #
3398        if [[ -f $nfile ]]; then
3399                ocr=$ofile
3400                [[ ! -f $ofile ]] && ocr=/dev/null

3402                if [[ -z $mv_but_nodiff ]]; then
3403                        textcomm=`getcomments text $P $PP`
3404                        if [[ -x $CODEREVIEW ]]; then
3405                                $CODEREVIEW -y "$textcomm" \
3406                                    -e $ocr $nfile \
3407                                    > /tmp/$$.psfile 2>/dev/null &&
3408                                    cat /tmp/$$.psfile >> $WDIR/$WNAME.ps
3409                                if [[ $? -eq 0 ]]; then
3410                                        print " ps\c"
3411                                else
3412                                        print " ps[fail]\c"
3413                                fi
3414                        fi
3415                fi
3416        fi

3418        if [[ -f $ofile ]]; then
3419                source_to_html Old $PP < $ofile > $WDIR/$DIR/$F-.html
3420                print " old\c"
3421        fi

3423        if [[ -f $nfile ]]; then
3424                source_to_html New $P < $nfile > $WDIR/$DIR/$F.html
3425                print " new\c"
3426        fi

3428        cd $OWD

3430        print
```



```
3431 done

3433 frame_nav_js > $WDIR/ancnav.js
3434 frame_navigation > $WDIR/ancnav.html

3436 if [[ ! -f $WDIR/$WNAME.ps ]]; then
3437         print " Generating PDF: Skipped: no output available"
3438 elif [[ -x $CODEREVIEW && -x $PS2PDF ]]; then
3439         print " Generating PDF: \c"
3440         fix_postscript $WDIR/$WNAME.ps | $PS2PDF - > $WDIR/$WNAME.pdf
3441         print "Done."
3442 else
3443         print " Generating PDF: Skipped: missing 'ps2pdf' or 'codereview'"
3444 fi

3446 # If we're in OpenSolaris mode and there's a closed dir under $WDIR,
3447 # delete it - prevent accidental publishing of closed source

3449 if [[ -n "$Oflag" ]]; then
3450         $FIND $WDIR -type d -name closed -exec /bin/rm -rf {} \;
3451 fi

3453 # Now build the index.html file that contains
3454 # links to the source files and their diffs.

3456 cd $CWS

3458 # Save total changed lines for Code Inspection.
3459 print "$TOTL" > $WDIR/TotalChangedLines

3461 print "     index.html: \c"
3462 INDEXFILE=$WDIR/index.html
3463 exec 3<&1                       # duplicate stdout to FD3.
3464 exec 1<&-                       # Close stdout.
3465 exec > $INDEXFILE               # Open stdout to index file.

3467 print "$HTML<head>$STDHEAD"
3468 print "<title>$WNAME</title>"
3469 print "</head>"
3470 print "<body id=\"SUNWwebrev\">"
3471 print "<div class=\"summary\">"
3472 print "<h2>Code Review for $WNAME</h2>"

3474 print "<table>"

3476 #
3477 # Get the preparer's name:
3478 #
3479 # If the SCM detected is Mercurial, and the configuration property
3480 # ui.username is available, use that, but be careful to properly escape
3481 # angle brackets (HTML syntax characters) in the email address.
3482 #
3483 # For git, use the user.name and user.email properties if they are set.
3484 #
3485 # Otherwise, use the current userid in the form "John Doe (jdoe)", but
3486 # to maintain compatibility with passwd(4), we must support '&' substitutions.
3487 #
3488 preparer=
3489 if [[ "$SCM_MODE" == mercurial ]]; then
3490         preparer=`hg showconfig ui.username 2>/dev/null`
3491 elif [[ "$SCM_MODE" == git ]]; then
3492         preparer=$( git config user.name 2>/dev/null )
3493         prepmail=$( git config user.email 2>/dev/null )
3494         if [[ -n "$prepmail" ]]; then
3495                 preparer="$preparer <$prepmail>"
3488         if [[ -n "$preparer" ]]; then
```

```
3489                 preparer="$(echo "$preparer" | html_quote)"
3496         fi
3497 fi
3498 if [[ -n "$preparer" ]]; then
3499         preparer="$(echo "$preparer" | html_quote)"
3500 fi
3501 if [[ -z "$preparer" ]]; then
3502         preparer=$(
3503             $PERL -e '
3504                 ($login, $pw, $uid, $gid, $quota, $cmt, $gcos) = getpwuid($<);
3505                 if ($login) {
3506                     $gcos =~ s/\&/ucfirst($login)/e;
3507                     printf "%s (%s)\n", $gcos, $login;
3508                 } else {
3509                     printf "(unknown)\n";
3510                 }
3511         ')
3512 fi

3514 PREPDATE=$(LC_ALL=C /usr/bin/date +%Y-%b-%d\ %R\ %z\ %Z)
3515 print "<tr><th>Prepared by:</th><td>$preparer on $PREPDATE</td></tr>"
3516 print "<tr><th>Workspace:</th><td>${PRETTY_CWS:-$CWS}"
3517 print "</td></tr>"
3518 print "<tr><th>Compare against:</th><td>"
3519 if [[ -n $parent_webrev ]]; then
3520         print "webrev at $parent_webrev"
3521 else
3522         print "${PRETTY_PWS:-$PWS}"
3523 fi
3524 print "</td></tr>"
3525 print "<tr><th>Summary of changes:</th><td>"
3526 printCI $TOTL $TINS $TDEL $TMOD $TUNC
3527 print "</td></tr>"

3529 if [[ -f $WDIR/$WNAME.patch ]]; then
3530         wpatch_url="$(print $WNAME.patch | url_encode)"
3531         print "<tr><th>Patch of changes:</th><td>"
3532         print "<a href=\"$wpatch_url\">$WNAME.patch</a></td></tr>"
3533 fi
3534 if [[ -f $WDIR/$WNAME.pdf ]]; then
3535         wpdf_url="$(print $WNAME.pdf | url_encode)"
3536         print "<tr><th>Printable review:</th><td>"
3537         print "<a href=\"$wpdf_url\">$WNAME.pdf</a></td></tr>"
3538 fi

3540 if [[ -n "$iflag" ]]; then
3541         print "<tr><th>Author comments:</th><td><div>"
3542         cat /tmp/$$.include
3543         print "</div></td></tr>"
3544 fi
3545 print "</table>"
3546 print "</div>"

3548 #
3549 # Second pass through the files: generate the rest of the index file
3550 #
3551 cat $FLIST | while read LINE
3552 do
3553         set - $LINE
3554         P=$1

3556         if [[ $# == 2 ]]; then
3557                 PP=$2
3558                 oldname="$PP"
3559         else
3560                 PP=$P
```



```
3561                 oldname=""
3562         fi

3564         mv_but_nodiff=
3565         cmp $WDIR/raw_files/old/$PP $WDIR/raw_files/new/$P > /dev/null 2>&1
3566         if [[ $? == 0 && -n "$oldname" ]]; then
3567                 mv_but_nodiff=1
3568         fi

3570         DIR=${P%/*}
3571         if [[ $DIR == $P ]]; then
3572                 DIR="."   # File at root of workspace
3573         fi

3575         # Avoid processing the same file twice.
3576         # It's possible for renamed files to
3577         # appear twice in the file list

3579         F=$WDIR/$P

3581         print "<p>"

3583         # If there's a diffs file, make diffs links

3585         if [[ -f $F.cdiff.html ]]; then
3586                 cdiff_url="$(print $P.cdiff.html | url_encode)"
3587                 udiff_url="$(print $P.udiff.html | url_encode)"
3588                 print "<a href=\"$cdiff_url\">Cdiffs</a>"
3589                 print "<a href=\"$udiff_url\">Udiffs</a>"

3591                 if [[ -f $F.wdiff.html && -x $WDIFF ]]; then
3592                         wdiff_url="$(print $P.wdiff.html | url_encode)"
3593                         print "<a href=\"$wdiff_url\">Wdiffs</a>"
3594                 fi

3596                 sdiff_url="$(print $P.sdiff.html | url_encode)"
3597                 print "<a href=\"$sdiff_url\">Sdiffs</a>"

3599                 frames_url="$(print $P.frames.html | url_encode)"
3600                 print "<a href=\"$frames_url\">Frames</a>"
3601         else
3602                 print " ------ ------ ------"

3604                 if [[ -x $WDIFF ]]; then
3605                         print " ------"
3606                 fi

3608                 print " ------"
3609         fi

3611         # If there's an old file, make the link

3613         if [[ -f $F-.html ]]; then
3614                 oldfile_url="$(print $P-.html | url_encode)"
3615                 print "<a href=\"$oldfile_url\">Old</a>"
3616         else
3617                 print " ---"
3618         fi

3620         # If there's an new file, make the link

3622         if [[ -f $F.html ]]; then
3623                 newfile_url="$(print $P.html | url_encode)"
3624                 print "<a href=\"$newfile_url\">New</a>"
3625         else
3626                 print " ---"
```

```
3627         fi

3629         if [[ -f $F.patch ]]; then
3630                 patch_url="$(print $P.patch | url_encode)"
3631                 print "<a href=\"$patch_url\">Patch</a>"
3632         else
3633                 print " -----"
3634         fi

3636         if [[ -f $WDIR/raw_files/new/$P ]]; then
3637                 rawfiles_url="$(print raw_files/new/$P | url_encode)"
3638                 print "<a href=\"$rawfiles_url\">Raw</a>"
3639         else
3640                 print " ---"
3641         fi

3643         print "<b>$P</b>"

3645         # For renamed files, clearly state whether or not they are modified
3646         if [[ -f "$oldname" ]]; then
3647                 if [[ -n "$mv_but_nodiff" ]]; then
3648                         print "<i>(copied from $oldname)</i>"
3649                 else
3650                         print "<i>(copied and modified from $oldname)</i>"
3651                 fi
3652         elif [[ -n "$oldname" ]]; then
3653                 if [[ -n "$mv_but_nodiff" ]]; then
3654                         print "<i>(renamed from $oldname)</i>"
3655                 else
3656                         print "<i>(renamed and modified from $oldname)</i>"
3657                 fi
3658         fi

3660         # If there's an old file, but no new file, the file was deleted
3661         if [[ -f $F-.html && ! -f $F.html ]]; then
3662                 print " <i>(deleted)</i>"
3663         fi

3665         #
3666         # Check for usr/closed and deleted_files/usr/closed
3667         #
3668         if [ ! -z "$Oflag" ]; then
3669                 if [[ $P == usr/closed/* || \
3670                     $P == deleted_files/usr/closed/* ]]; then
3671                         print "  <i>Closed source: omitted from" \
3672                             "this review</i>"
3673                 fi
3674         fi

3676         print "</p>"
3677         # Insert delta comments

3679         print "<blockquote><pre>"
3680         getcomments html $P $PP
3681         print "</pre>"

3683         # Add additional comments comment

3685         print "<!-- Add comments to explain changes in $P here -->"

3687         # Add count of changes.

3689         if [[ -f $F.count ]]; then
3690             cat $F.count
3691             rm $F.count
3692         fi
```



```
3694         if [[ $SCM_MODE == "teamware" ||
3695             $SCM_MODE == "mercurial" ||
3696             $SCM_MODE == "unknown" ]]; then

3698                 # Include warnings for important file mode situations:
3699                 # 1) New executable files
3700                 # 2) Permission changes of any kind
3701                 # 3) Existing executable files

3703                 old_mode=
3704                 if [[ -f $WDIR/raw_files/old/$PP ]]; then
3705                         old_mode=‘get_file_mode $WDIR/raw_files/old/$PP‘
3706                 fi

3708                 new_mode=
3709                 if [[ -f $WDIR/raw_files/new/$P ]]; then
3710                         new_mode=‘get_file_mode $WDIR/raw_files/new/$P‘
3711                 fi

3713                 if [[ -z "$old_mode" && "$new_mode" = *[1357]* ]]; then
3714                         print "<span class=\"chmod\">"
3715                         print "<p>new executable file: mode $new_mode</p>"
3716                         print "</span>"
3717                 elif [[ -n "$old_mode" && -n "$new_mode" &&
3718                     "$old_mode" != "$new_mode" ]]; then
3719                         print "<span class=\"chmod\">"
3720                         print "<p>mode change: $old_mode to $new_mode</p>"
3721                         print "</span>"
3722                 elif [[ "$new_mode" = *[1357]* ]]; then
3723                         print "<span class=\"chmod\">"
3724                         print "<p>executable file: mode $new_mode</p>"
3725                         print "</span>"
3726                 fi
3727         fi

3729         print "</blockquote>"
3730 done

3732 print
3733 print
3734 print "<hr></hr>"
3735 print "<p style=\"font-size: small\">"
3736 print "This code review page was prepared using <b>$0</b>."
3737 print "Webrev is maintained by the <a href=\"http://www.illumos.org\">"
3738 print "illumos</a> project.  The latest version may be obtained"
3739 print "<a href=\"http://src.illumos.org/source/xref/illumos-gate/usr/src/tools/s
3740 print "</body>"
3741 print "</html>"

3743 exec 1<&-                       # Close FD 1.
3744 exec 1<&3                       # dup FD 3 to restore stdout.
3745 exec 3<&-                       # close FD 3.

3747 print "Done."

3749 #
3750 # If remote deletion was specified and fails do not continue.
3751 #
3752 if [[ -n $Dflag ]]; then
3753         delete_webrev 1 1
3754         (( $? == 0 )) || exit $?
3755 fi

3757 if [[ -n $Uflag ]]; then
3758         upload_webrev
```

```
3759         exit $?
3760 fi
```