

```
new/usr/src/tools/scripts/webrev.sh
```

```
*****
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
*****
1 #!/usr/bin/ksh93 -p
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #

23 #
24 # Copyright (c) 2002, 2010, Oracle and/or its affiliates. All rights reserved.
25 #

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

31 #
32 # This script takes a file list and a workspace and builds a set of html files
33 # suitable for doing a code review of source changes via a web page.
34 # Documentation is available via the manual page, webrev.1, or just
35 # type 'webrev -h'.
36 #
37 # Acknowledgements to contributors to webrev are listed in the webrev(1)
38 # man page.
39 #

41 REMOVED_COLOR=brown
42 CHANGED_COLOR=blue
43 NEW_COLOR=blue

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

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

55 STDHEAD='<meta http-equiv="cache-control" content="no-cache"></meta>
56 <meta http-equiv="Pragma" content="no-cache"></meta>
57 <meta http-equiv="Expires" content="-1"></meta>
58 <!--
59 Note to customizers: the body of the webrev is IDed as SUNWwebrev
60 to allow easy overriding by users of webrev via the userContent.css
61 mechanism available in some browsers.
```

1

```
new/usr/src/tools/scripts/webrev.sh
```

```
63     For example, to have all "removed" information be red instead of
64 brown, set a rule in your userContent.css file like:
65
66         body#SUNWwebrev span.removed { color: red ! important; }
67 -->
68 <style type="text/css" media="screen">
69 body {
70     background-color: #eeeeee;
71 } unchanged_portion_omitted

927 #
928 # fix_postscript
929 #

2260 #
2261 #
2262 # Main program starts here
2263 #
2264 #

2266 trap "rm -f /tmp/$$.* ; exit" 0 1 2 3 15
2268 set +o noclobber

2270 PATH=$(/bin dirname "$(whence $0)":$PATH
2272 [[ -z $WDIFF ]] && WDIFF='look_for_prog wdiff'
2273 [[ -z $WX ]] && WX='look_for_prog wx'
2274 [[ -z $HG_ACTIVE ]] && HG_ACTIVE='look_for_prog hg-active'
2275 [[ -z $GIT ]] && GIT='look_for_prog git'
2276 [[ -z $WHICH_SCM ]] && WHICH_SCM='look_for_prog which_scm'
2277 [[ -z $CODEREVIEW ]] && CODEREVIEW='look_for_prog codereview'
2278 [[ -z $PS2PDF ]] && PS2PDF='look_for_prog ps2pdf'
2279 [[ -z $PERL ]] && PERL='look_for_prog perl'
2280 [[ -z $RSYNC ]] && RSYNC='look_for_prog rsync'
2281 [[ -z $SCCS ]] && SCCS='look_for_prog sccs'
2282 [[ -z $AWK ]] && AWK='look_for_prog nawk'
2283 [[ -z $AWK ]] && AWK='look_for_prog gawk'
2284 [[ -z $AWK ]] && AWK='look_for_prog awk'
2285 [[ -z $SCP ]] && SCP='look_for_prog scp'
2286 [[ -z $SED ]] && SED='look_for_prog sed'
2287 [[ -z $SFTP ]] && SFTP='look_for_prog sftp'
2288 [[ -z $SORT ]] && SORT='look_for_prog sort'
2289 [[ -z $MKTEMP ]] && MKTEMP='look_for_prog mktemp'
2290 [[ -z $GREP ]] && GREP='look_for_prog grep'
2291 [[ -z $FIND ]] && FIND='look_for_prog find'

2293 # set name of trash directory for remote webrev deletion
2294 TRASH_DIR=".trash"
2295 [[ -n $WEBREV_TRASH_DIR ]] && TRASH_DIR=$WEBREV_TRASH_DIR

2297 if [[ ! -x $PERL ]]; then
2298     print -u2 "Error: No perl interpreter found. Exiting."
2299     exit 1
2300 fi

2302 if [[ ! -x $WHICH_SCM ]]; then
2303     print -u2 "Error: Could not find which_scm. Exiting."
2304     exit 1
2305 fi

2307 #
2308 # These aren't fatal, but we want to note them to the user.
```

2

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

```
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:D:i: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=$OPTARG;;
2358         I)      Iflag=1
2359             ITSRG=$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;;
```

3

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

```
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 $codemgr_ws ]] && codemgr_ws=$(cd $codemgr_ws; print $PWD)
2430     codemgr_ws=$($codemgr_ws)
2431     CODEMGR_WS=$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     #
```

4

```

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
2492     rel=$url#$repo
2493     CWS=${PWD%$rel}
2494 fi
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)

```

```

2706 elif [[ $SCM_MODE == "mercurial" ]]; then
2707     #
2708     # Parent can either be specified with -p
2709     # Specified with CODEMGR_PARENT in the environment
2710     # or taken from hg's default path.
2711     #
2712
2713     if [[ -z $codemgr_parent && -n $CODEMGR_PARENT ]]; then
2714         codemgr_parent=$CODEMGR_PARENT
2715     fi
2716
2717     if [[ -z $codemgr_parent ]]; then
2718         codemgr_parent='hg path -R $codemgr_ws default 2>/dev/null'
2719     fi
2720
2721     PWS=$codemgr_parent
2722
2723     #
2724     # If the parent is a webrev, we want to do some things against
2725     # the natural workspace parent (file list, comments, etc)
2726     #
2727     if [[ -n $parent_webrev ]]; then
2728         real_parent=$(hg path -R $codemgr_ws default 2>/dev/null)
2729     else
2730         real_parent=$PWS
2731     fi
2732
2733     #
2734     # If hg-active exists, then we run it. In the case of no explicit
2735     # flist given, we'll use it for our comments. In the case of an
2736     # explicit flist given we'll try to use it for comments for any
2737     # files mentioned in the flist.
2738     #
2739     if [[ -z $flist_done ]]; then
2740         flist_from_mercurial $CWS $real_parent
2741         flist_done=1
2742     fi
2743
2744     #
2745     # If we have a file list now, pull out any variables set
2746     # therein. We do this now (rather than when we possibly use
2747     # hg-active to find comments) to avoid stomping specifications
2748     # in the user-specified flist.
2749     #
2750     if [[ -n $flist_done ]]; then
2751         env_from_flist
2752     fi
2753
2754     #
2755     # Only call hg-active if we don't have a wx formatted file already
2756     #
2757     if [[ -x $HG_ACTIVE && -z $wxfile ]]; then
2758         print " Comments from: hg-active -p $real_parent ... \c"
2759         hg_active_wxfile $CWS $real_parent
2760         print " Done."
2761     fi
2762
2763     #
2764     # At this point we must have a wx flist either from hg-active,
2765     # or in general. Use it to try and find our parent revision,
2766     # if we don't have one.
2767     #
2768     if [[ -z $HG_PARENT ]]; then
2769         eval '$SED -e "s/#.*$/ /' $wxfile | $GREP HG_PARENT='
2770     fi

```

```

2772     #
2773     # If we still don't have a parent, we must have been given a
2774     # wx-style active list with no HG_PARENT specification, run
2775     # hg-active and pull an HG_PARENT out of it, ignore the rest.
2776     #
2777     if [[ -z $HG_PARENT && -x $HG_ACTIVE ]]; then
2778         $HG_ACTIVE -w $codemgr_ws -p $real_parent | \
2779             eval '$SED -e "s/#.*$/ /' | $GREP HG_PARENT='
2780     elif [[ -z $HG_PARENT ]]; then
2781         print -u2 "Error: Cannot discover parent revision"
2782         exit 1
2783     fi
2784
2785     pnode=$(trim_digest $HG_PARENT)
2786     PRETTY_PWS="${PWS} (at ${pnode})"
2787     cnode=$(hg parent -R $codemgr_ws --template '{node|short}' \
2788             2>/dev/null)
2789     PRETTY_CWS="${CWS} (at ${cnode})"
2790     elif [[ $SCM_MODE == "git" ]]; then
2791     #
2792     # Parent can either be specified with -p, or specified with
2793     # CODEMGR_PARENT in the environment.
2794     #
2795     if [[ -z $codemgr_parent && -n $CODEMGR_PARENT ]]; then
2796         codemgr_parent=$CODEMGR_PARENT
2797     fi
2798
2799     # Try to figure out the parent based on the branch the current
2800     # branch is tracking, if we fail, use origin/master
2801     this_branch=$(GIT branch | awk '$1 == "*" { print $2 }')
2802     par_branch="origin/master"
2803
2804     # If we're not on a branch there's nothing we can do
2805     if [[ $this_branch != "(no branch)" ]]; then
2806         $GIT for-each-ref
2807             --format='%(refname:short) %(upstream:short)' refs/heads/ |
2808             while read local remote; do
2809                 [[ "$local" == "$this_branch" ]] && par_branch="$remote"
2810             done
2811     fi
2812
2813     if [[ -z $codemgr_parent ]]; then
2814         codemgr_parent=$par_branch
2815     fi
2816
2817     PWS=$codemgr_parent
2818
2819     #
2820     # If the parent is a webrev, we want to do some things against
2821     # the natural workspace parent (file list, comments, etc)
2822     #
2823     if [[ -n $parent_webrev ]]; then
2824         real_parent=$par_branch
2825     else
2826         real_parent=$PWS
2827     fi
2828
2829     if [[ -z $flist_done ]]; then
2830         flist_from_git "$CWS" "$real_parent"
2831         flist_done=1
2832     fi
2833
2834     #
2835     # If we have a file list now, pull out any variables set
2836     # therein.

```

```

2837      #
2838      if [[ -n $flist_done ]]; then
2839          env_from_flist
2840      fi
2842
2843      #
2844      # If we don't have a wx-format file list, build one we can pull change
2845      # comments from.
2846      #
2847      if [[ -z $wxfile ]]; then
2848          print " Comments from: git...\\c"
2849          git_wxfile "$CWS" "$real_parent"
2850          print " Done."
2851      fi
2852
2853      if [[ -z $GIT_PARENT ]]; then
2854          GIT_PARENT=$(git merge-base "$real_parent" HEAD)
2855      fi
2856      if [[ -z $GIT_PARENT ]]; then
2857          print -u2 "Error: Cannot discover parent revision"
2858          exit 1
2859      fi
2860
2861      pnode=$(trim_digest $GIT_PARENT)
2862
2863      if [[ $real_parent == /*/* ]]; then
2864          origin=$(echo $real_parent | cut -d/ -f1)
2865          origin=$($GIT remote -v | \
2866          $AWK '$1 == "'$origin'" { print $2; exit }')
2867      else
2868          PRETTY_PWS="${PWS} (at ${pnode})"
2869      fi
2870
2871      cnode=$(($GIT --git-dir=${codemgr_ws}/.git rev-parse --short=12 HEAD \
2872      2>/dev/null)
2873      PRETTY_CWS="$CWS (at ${cnode})"
2874  elif [[ $SCM_MODE == "subversion" ]]; then
2875
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
2881
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
2891
2892      env_from_flist
2893
2894      if [[ -z $CODEMGR_WS ]]; then
2895          print -u2 "SCM not detected/supported and CODEMGR_WS not specified"
2896          exit 1
2897      fi
2898
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      else
2922          #
2923          # $INCLUDE_FILE may be a relative path, and the script alters
2924          # PWD, so we just stash a copy in /tmp.
2925          #
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 $nflag ]]; then
2940      DEFREGFILE=$(dirname "$(whence $0)"/..etc/its.reg"
2941      DEFREGFILE0=$(dirname "$(whence $0)"/its.reg"
2942      if [[ -n $iflag ]]; then
2943          REGFILE=$ITSREG
2944      elif [[ -r $HOME/.its.reg ]]; then
2945          REGFILE=$HOME/.its.reg
2946      elif [[ -r $DEFREGFILE ]]; then
2947          REGFILE=$DEFREGFILE
2948      else
2949          REGFILE=$DEFREGFILE0
2950          REGFILE=$DEFREGFILE
2951      fi
2952      if [[ ! -r $REGFILE ]]; then
2953          print "ERROR: Unable to read database registry file $REGFILE"
2954          exit 1
2955      elif [[ $REGFILE != $DEFREGFILE ]]; then
2956          print " its.reg from: $REGFILE"
2957      fi
2958      $SED -e '/^#/d' -e '/^[' ]*$/d' $REGFILE | while read LINE; do
2959          name=${LINE%%=*}
2960          value=${LINE##*=}"
2961          if [[ $name == PREFIX ]]; then
2962              p=$value
2963              valid_prefixes="${p} ${valid_prefixes}"
2964          else
2965              itsinfo["${p}_${name}"]="$value"
2966          fi
2967      done
2968  fi

```

```

2968         fi
2969     done

2972     DEFCONFFILE=$(/bin dirname "$(whence $0)"/etc/its.conf"
2973     DEFCONFFILE0=$(/bin dirname "$(whence $0)"/its.conf"
2974     if [[ -r $DEFCONFFILE ]]; then
2975         CONFFILES=$DEFCONFFILE
2976     else
2977         CONFFILES=$DEFCONFFILE0
2978     fi
2979     if [[ -r ${HOME}/.its.conf ]]; then
2980         CONFFILES="${CONFFILES} ${HOME}/.its.conf"
2981     fi
2982     if [[ -n $Cflag ]]; then
2983         CONFFILES="${CONFFILES} ${ITSCONF}"
2984     fi
2985     its_domain=
2986     its_priority=
2987     for cf in ${CONFFILES}; do
2988         if [[ ! -r $cf ]]; then
2989             print "ERROR: Unable to read database configuration file"
2990             exit 1
2991         elif [[ $cf != $DEFCONFFILE ]]; then
2992             print "           its.conf: reading $cf"
2993         fi
2994         $SED -e '/^#/d' -e '/^[[      ]]*$/d' $cf | while read LINE; do
2995             eval "$LINE"
2996         done
2997     done

2998     #
2999     # If an information tracking system is explicitly identified by prefix,
3000     # we want to disregard the specified priorities and resolve it according
3001     #
3002     # To that end, we'll build a sed script to do each valid prefix in turn.
3003     #
3004     for p in ${valid_prefixes}; do
3005         #
3006         # When an informational URL was provided, translate it to a
3007         # hyperlink. When omitted, simply use the prefix text.
3008         #
3009         if [[ -z ${itsinfo["${p}_INFO"]} ]]; then
3010             itsinfo["${p}_INFO"]=${p}
3011         else
3012             itsinfo["${p}_INFO"]="

```

```

3034         fi

3036         #
3037         # Turn the destination URL into a hyperlink
3038         #
3039         itsinfo["${p}_URL"]="\">\${itsinfo\["\${p}\_URL"\]}">&</a>"

3041         # The character class below contains a literal tab
3042         print "/^${p}[:          ]/ {"
3043             s:${itsinfo["${p}_REGEX"]};${itsinfo["${p}_URL"]};g
3044             s;^${p};${itsinfo["${p}_INFO"]};g
3045         }" >> ${its_sed_script}
3046     done

3048     #
3049     # The previous loop took care of explicit specification. Now use
3050     # the configured priorities to attempt implicit translations.
3051     #
3052     for p in ${its_priority}; do
3053         print "/^${itsinfo["${p}_REGEX"]}[          ]/ {"
3054             s;^${itsinfo["${p}_REGEX"]};${itsinfo["${p}_URL"]};g
3055         }" >> ${its_sed_script}
3056     done
3057 fi

3059 #
3060 # Search for DO_EVERYTHING above for matching "for" statement
3061 # and explanation of this terminator.
3062 #
3063 done

3065 #
3066 # Output directory.
3067 #
3068 WDIR=${WDIR:-$CWS/webrev}

3070 #
3071 # Name of the webrev, derived from the workspace name or output directory;
3072 # in the future this could potentially be an option.
3073 #
3074 if [[ -n $oflag ]]; then
3075     WNAME=${WDIR##*/}
3076 else
3077     WNAME=${CWS##*/}
3078 fi

3079 # Make sure remote target is well formed for remote upload/delete.
3080 if [[ -n $dflag || -n $uflag ]]; then
3081     #
3082         # If remote target is not specified, build it from scratch using
3083         # the default values.
3084         #
3085         #
3086         if [[ -z $tflag ]]; then
3087             remote_target=${DEFAULT_REMOTE_HOST}:${WNAME}
3088         else
3089             #
3090             # Check upload target prefix first.
3091             #
3092             if [[ "${remote_target}" != ${rsync_prefix}* &&
3093                  "${remote_target}" != ${ssh_prefix}* ]]; then
3094                 print "ERROR: invalid prefix of upload URI" \
3095                         "($remote_target)"
3096                 exit 1
3097             fi
3098             #
3099             # If destination specification is not in the form of

```

```

3100 # host_spec:remote_dir then assume it is just remote hostname
3101 # and append a colon and destination directory formed from
3102 # local webrev directory name.
3103 #
3104 typeset target_no_prefix=${remote_target##*:}
3105 if [[ $target_no_prefix == *:* ]]; then
3106     if [[ ${remote_target} == *: ]]; then
3107         remote_target=${remote_target}${WNAME}
3108     fi
3109     else
3110         if [[ ${target_no_prefix} == */* ]]; then
3111             print "ERROR: badly formed upload URI" \
3112                 "($remote_target)"
3113             exit 1
3114         else
3115             remote_target=${remote_target}:${WNAME}
3116         fi
3117     fi
3118 fi
3119 #
3120 # Strip trailing slash. Each upload method will deal with directory
3121 # specification separately.
3122 #
3123 #
3124 remote_target=${remote_target%/}
3125 fi

3126 #
3127 # Option -D by itself (option -U not present) implies no webrev generation.
3128 #
3129 if [[ -z $Uflag && -n $Dflag ]]; then
3130     delete_webrev 1 1
3131     exit $?
3133 fi

3134 #
3135 # Do not generate the webrev, just upload it or delete it.
3136 #
3137 #
3138 if [[ -n $nflag ]]; then
3139     if [[ -n $Dflag ]]; then
3140         delete_webrev 1 1
3141         (( $? == 0 )) || exit $?
3142     fi
3143     if [[ -n $Uflag ]]; then
3144         upload_webrev
3145         exit $?
3146     fi
3147 fi

3148 if [ "${WDIR%%/*}" ]; then
3149     WDIR=$PWD/$WDIR
3150 fi

3151 if [[ ! -d $WDIR ]]; then
3152     mkdir -p $WDIR
3153     (( $? != 0 )) && exit 1
3156 fi

3157 #
3158 # Summarize what we're going to do.
3159 #
3160 print "    Workspace: ${PRETTY_CWS:-$CWS}"
3161 if [[ -n $parent_webrev ]]; then
3162     print "Compare against: webrev at $parent_webrev"
3164 else
3165     print "Compare against: ${PRETTY_PWS:-$PWS}"

```

```

3166 fi
3168 [[ -n $INCLUDE_FILE ]] && print "      Including: $INCLUDE_FILE"
3169 print "          Output to: $WDIR"
3171 #
3172 # Save the file list in the webrev dir
3173 #
3174 [[ ! $FLIST -ef $WDIR/file.list ]] && cp $FLIST $WDIR/file.list
3176 rm -f $WDIR/$WNAME.patch
3177 rm -f $WDIR/$WNAME.ps
3178 rm -f $WDIR/$WNAME.pdf
3180 touch $WDIR/$WNAME.patch
3182 print "      Output Files:"
3184 #
3185 # Clean up the file list: Remove comments, blank lines and env variables.
3186 #
3187 SSED -e "s/#.*//g" -e "/=d" -e "/^[\t]*$/d" $FLIST > /tmp/$$.list.clean
3188 FLIST=/tmp/$$.list.clean
3190 #
3191 # For Mercurial, create a cache of manifest entries.
3192 #
3193 if [[ $SCM_MODE == "mercurial" ]]; then
3194     #
3195     # Transform the FLIST into a temporary sed script that matches
3196     # relevant entries in the Mercurial manifest as follows:
3197     # 1) The script will be used against the parent revision manifest,
3198     # so for FLIST lines that have two filenames (a renamed file)
3199     # keep only the old name.
3200     # 2) Escape all forward slashes the filename.
3201     # 3) Change the filename into another sed command that matches
3202     # that file in "hg manifest -v" output: start of line, three
3203     # octal digits for file permissions, space, a file type flag
3204     # character, space, the filename, end of line.
3205     # 4) Eliminate any duplicate entries. (This can occur if a
3206     # file has been used as the source of an hg cp and it's
3207     # also been modified in the same changeset.)
3208     #
3209     SEDFILE=/tmp/$$.manifest.sed
3210     $SED '
3211     $S#[^\^]*##g
3212     $S#[\\\^]/#g
3213     $S#[^\^]*$##g
3214     ' < $FLIST | $SORT -u > $SEDFILE
3216 #
3217 # Apply the generated script to the output of "hg manifest -v"
3218 # to get the relevant subset for this webrev.
3219 #
3220 HG_PARENT_MANIFEST=/tmp/$$.manifest
3221 hg -R $CWS manifest -v -r $HG_PARENT |
3222     $SED -n -f $SEDFILE > $HG_PARENT_MANIFEST
3223 fi

3225 #
3226 # First pass through the files: generate the per-file webrev HTML-files.
3227 #
3228 cat $FLIST | while read LINE
3229 do
3230     set - $LINE
3231     P=$1

```

```

3233      #
3234      # Normally, each line in the file list is just a pathname of a
3235      # file that has been modified or created in the child. A file
3236      # that is renamed in the child workspace has two names on the
3237      # line: new name followed by the old name.
3238      #
3239      oldname=""
3240      oldpath=""
3241      rename=
3242      if [[ $# -eq 2 ]]; then
3243          PP=$2                      # old filename
3244          if [[ -f $PP ]]; then
3245              oldname="(copied from $PP)"
3246          else
3247              oldname="(renamed from $PP)"
3248          fi
3249          oldpath="$PP"
3250          rename=1
3251          PDIR=${PP%/*}
3252          if [[ $PDIR == $PP ]]; then
3253              PDIR=". "                 # File at root of workspace
3254          fi
3255          PF=${P##*/}
3256          DIR=${P%/*}
3257          if [[ $DIR == $P ]]; then
3258              DIR=". "                 # File at root of workspace
3259          fi
3260          F=${P##*/}
3261
3262      else
3263          DIR=${P%/*}
3264          if [[ "$DIR" == "$P" ]]; then
3265              DIR=". "                 # File at root of workspace
3266          fi
3267          F=${P##*/}
3268          PP=$P
3269          PDIR=$DIR
3270          PF=$F
3271      fi
3272
3273      COMM='getcomments html $P $PP'
3274
3275      print "\t$P$oldname\n\t\t\c"
3276
3277      # Make the webrev mirror directory if necessary
3278      mkdir -p $WDIR/$DIR
3279
3280      #
3281      # We stash old and new files into parallel directories in $WDIR
3282      # and do our diffs there. This makes it possible to generate
3283      # clean looking diffs which don't have absolute paths present.
3284      #
3285      build_old_new "$WDIR" "$PWS" "$PDIR" "$PF" "$CWS" "$DIR" "$F" || \
3286          continue
3287
3288      #
3289      # Keep the old PWD around, so we can safely switch back after
3290      # diff generation, such that build_old_new runs in a
3291      # consistent environment.

```

```

3298      #
3299      OWD=$PWD
3300      cd $WDIR/raw_files
3301      ofile=old/$PDIR/$PF
3302      nfile=new/$DIR/$F
3303
3304      mv_but_nodiff=
3305      cmp $ofile $nfile > /dev/null 2>&1
3306      if [[ $? == 0 && $rename == 1 ]]; then
3307          mv_but_nodiff=1
3308      fi
3309
3310      #
3311      # If we have old and new versions of the file then run the appropriate
3312      # diffs. This is complicated by a couple of factors:
3313      #
3314      # - renames must be handled specially: we emit a 'remove'
3315      #   diff and an 'add' diff
3316      # - new files and deleted files must be handled specially
3317      # - Solaris patch(lm) can't cope with file creation
3318      #   (and hence renames) as of this writing.
3319      # - To make matters worse, gnu patch doesn't interpret the
3320      #   output of Solaris diff properly when it comes to
3321      #   adds and deletes. We need to do some "cleansing"
3322      #   transformations:
3323      #       [to add a file] @@ -1,0 +X,Y @@ --> @@ -0,0 +X,Y @@
3324      #       [to del a file] @@ -X,Y +1,0 @@ --> @@ -X,Y +0,0 @@
3325      #
3326      cleanse_rmfile="$SED 's/^\\(@@ [0-9+, -]*\\) [0-9+, -]* @@$/\\1 +0,0 @@/'"
3327      cleanse_newfile="$SED 's/^@@ [0-9+, -]* \\([0-9+, -]* @@\\)$/@@ -0,0 \\1/'"
3328
3329      rm -f $WDIR/$DIR/$F.patch
3330      if [[ -z $rename ]]; then
3331          if [ ! -f "$ofile" ]; then
3332              diff -u /dev/null $nfile | sh -c "$cleanse_newfile" \
3333                  > $WDIR/$DIR/$F.patch
3334          elif [ ! -f "$nfile" ]; then
3335              diff -u $ofile /dev/null | sh -c "$cleanse_rmfile" \
3336                  > $WDIR/$DIR/$F.patch
3337          else
3338              diff -u $ofile $nfile > $WDIR/$DIR/$F.patch
3339          fi
3340      else
3341          diff -u $ofile /dev/null | sh -c "$cleanse_rmfile" \
3342                  > $WDIR/$DIR/$F.patch
3343
3344          diff -u /dev/null $nfile | sh -c "$cleanse_newfile" \
3345                  >> $WDIR/$DIR/$F.patch
3346      fi
3347
3348      #
3349      # Tack the patch we just made onto the accumulated patch for the
3350      # whole wad.
3351      #
3352      cat $WDIR/$DIR/$F.patch >> $WDIR/$WNAME.patch
3353
3354      print " patch\c"
3355
3356      if [[ -f $ofile && -f $nfile && -z $mv_but_nodiff ]]; then
3357          ${CDIFFCMD:-diff -bt -C 5} $ofile $nfile > $WDIR/$DIR/$F.cdiff
3358          diff_to_html $F $DIR/$F "C" "$COMM" < $WDIR/$DIR/$F.cdiff.html \
3359                  > $WDIR/$DIR/$F.cdiff.html
3360          print " cdiffs\c"
3361
3362      ${UDIFFCMD:-diff -bt -U 5} $ofile $nfile > $WDIR/$DIR/$F.udiff

```

```

3364     diff_to_html $F $DIR/$F "U" "$COMM" < $WDIR/$DIR/$F.udiff \
3365         > $WDIR/$DIR/$F.udiff.html
3367     print " udiffs\c"
3369
3370     if [[ -x $WDIFF ]]; then
3371         $WDIFF -c "$COMM" \
3372             -t "$WNAME Wdiff $DIR/$F" $ofile $nfile > \
3373                 $WDIR/$DIR/$F.wdiff.html 2>/dev/null
3374     if [[ $? -eq 0 ]]; then
3375         print " wdiffs\c"
3376     else
3377         print " wdiffs[fail]\c"
3378     fi
3380
3381     sdiff_to_html $ofile $nfile $F $DIR "$COMM" \
3382         > $WDIR/$DIR/$F.sdiff.html
3383     print " sdiffs\c"
3384
3385     print " frames\c"
3386
3387     rm -f $WDIR/$DIR/$F.cdiff $WDIR/$DIR/$F.udiff
3388
3389     difflines $ofile $nfile > $WDIR/$DIR/$F.count
3390
3391     elif [[ -f $ofile && -f $nfile && -n $mv_but_nodiff ]]; then
3392         # renamed file: may also have differences
3393         difflines $ofile $nfile > $WDIR/$DIR/$F.count
3394
3395     elif [[ -f $nfile ]]; then
3396         # new file: count added lines
3397         difflines /dev/null $nfile > $WDIR/$DIR/$F.count
3398
3399     elif [[ -f $ofile ]]; then
3400         # old file: count deleted lines
3401         difflines $ofile /dev/null > $WDIR/$DIR/$F.count
3402
3403     #
3404     # Now we generate the postscript for this file. We generate diffs
3405     # only in the event that there is delta, or the file is new (it seems
3406     # tree-killing to print out the contents of deleted files).
3407
3408     if [[ -f $nfile ]]; then
3409         ocr=$ofile
3410         [[ ! -f $ofile ]] && ocr=/dev/null
3411
3412         if [[ -z $mv_but_nodiff ]]; then
3413             textcomm='getcomments text $P $PP'
3414             if [[ -x $CODEREVIEW ]]; then
3415                 $CODEREVIEW -y "$textcomm" \
3416                     -e $ocr $nfile \
3417                         > /tmp/$$.psfile 2>/dev/null &&
3418                         cat /tmp/$$.psfile >> $WDIR/$WNAME.ps
3419             if [[ $? -eq 0 ]]; then
3420                 print " ps\c"
3421             else
3422                 print " ps[fail]\c"
3423             fi
3424         fi
3425
3426     if [[ -f $ofile ]]; then
3427         source_to_html Old $PP < $ofile > $WDIR/$DIR/$F-.html
3428         print " old\c"
3429     fi

```

```

3431         if [[ -f $nfile ]]; then
3432             source_to_html New $P < $nfile > $WDIR/$DIR/$F.html
3433             print " new\c"
3434         fi
3435
3436         cd $OWD
3437
3438         print
3439     done
3440
3441     frame_nav_js > $WDIR/ancnav.js
3442     frame_navigation > $WDIR/ancnav.html
3443
3444     if [[ ! -f $WDIR/$WNAME.ps ]]; then
3445         print " Generating PDF: Skipped: no output available"
3446     elif [[ -x $CODEREVIEW && -x $PS2PDF ]]; then
3447         print " Generating PDF: \c"
3448         fix_postscript $WDIR/$WNAME.ps | $PS2PDF - > $WDIR/$WNAME.pdf
3449         print "Done."
3450     else
3451         print " Generating PDF: Skipped: missing 'ps2pdf' or 'codereview'"
3452     fi
3453
3454     # If we're in OpenSolaris mode and there's a closed dir under $WDIR,
3455     # delete it - prevent accidental publishing of closed source
3456
3457     if [[ -n "$Oflag" ]]; then
3458         $FIND $WDIR -type d -name closed -exec /bin/rm -rf {} \;
3459     fi
3460
3461     # Now build the index.html file that contains
3462     # links to the source files and their diffs.
3463
3464     cd $CWS
3465
3466     # Save total changed lines for Code Inspection.
3467     print "$TOTL" > $WDIR/TotalChangedLines
3468
3469     print "    index.html: \c"
3470     INDEXFILE=$WDIR/index.html
3471     exec 3<&1
3472     exec 1<&-
3473     exec > $INDEXFILE
3474
3475     print "$HTML<head>$STDHEAD"
3476     print "<title>$WNAME</title>"
3477     print "</head>"
3478     print "<body id=\"SUNWwebrev\">"
3479     print "<div class=\"summary\">"
3480     print "<h2>Code Review for $WNAME</h2>"
3481
3482     print "<table>"
3483
3484     #
3485     # Get the preparer's name:
3486     #
3487     # If the SCM detected is Mercurial, and the configuration property
3488     # ui.username is available, use that, but be careful to properly escape
3489     # angle brackets (HTML syntax characters) in the email address.
3490     #
3491     # Otherwise, use the current userid in the form "John Doe (jdoe)", but
3492     # to maintain compatibility with passwd(4), we must support '&' substitutions.
3493     #
3494     preparer=
3495     if [[ "$SCM_MODE" == mercurial ]]; then

```

```

3496     preparer='hg showconfig ui.username 2>/dev/null'
3497     if [[ -n "$preparer" ]]; then
3498         preparer=$(echo "$preparer" | html_quote)"
3499     fi
3500 fi
3501 if [[ -z "$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, $login;
3508             } else {
3509                 printf "(unknown)\n";
3510             }
3511         ')
3512 fi
3513 PREPDATE=$(LC_ALL=C /usr/bin/date +%Y-%b-%d\ %R\ %z\ %Z)
3514 print "<tr><th>Prepared by:</th><td>$preparer on $PREPDATE</td></tr>"
3515 print "<tr><th>Workspace:</th><td>${PRETTY_CWS:-$CWS}"
3516 print "</td></tr>"
3517 print "<tr><th>Compare against:</th><td>""
3518 if [[ -n $parent_webrev ]]; then
3519     print "webrev at $parent_webrev"
3520 else
3521     print "${PRETTY_PWS:-$PWS}"
3522 fi
3523 print "</td></tr>""
3524 print "<tr><th>Summary of changes:</th><td>""
3525 printCI $TOTAL $TINS $TDEL $TMOD $TUNC
3526 print "</td></tr>""
3527
3528 if [[ -f $WDIR/$WNAME.patch ]]; then
3529     wpatch_url="$(print $WNAME.patch | url_encode)"
3530     print "<tr><th>Patch of changes:</th><td>""
3531     print "<a href=\"$wpatch_url\">$WNAME.patch</a></td></tr>""
3532 fi
3533 if [[ -f $WDIR/$WNAME.pdf ]]; then
3534     wpdf_url="$(print $WNAME.pdf | url_encode)"
3535     print "<tr><th>Printable review:</th><td>""
3536     print "<a href=\"$wpdf_url\">$WNAME.pdf</a></td></tr>""
3537 fi
3538
3539 if [[ -n "$iflag" ]]; then
3540     print "<tr><th>Author comments:</th><td><div>""
3541     cat /tmp/$$.include
3542     print "</div></td></tr>""
3543 fi
3544 print "</table>""
3545 print "</div>""
3546
3547 #
3548 # Second pass through the files: generate the rest of the index file
3549 #
3550 #
3551 cat $FLIST | while read LINE
3552 do
3553     set - $LINE
3554     P=$1
3555
3556     if [[ $# == 2 ]]; then
3557         PP=$2
3558         oldname="$PP"
3559     else
3560         PP=$P
3561         oldname=""

```

```

3562     fi
3563
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
3569
3570     DIR=${P%/*}
3571     if [[ $DIR == $P ]]; then
3572         DIR=." # File at root of workspace
3573     fi
3574
3575     # Avoid processing the same file twice.
3576     # It's possible for renamed files to
3577     # appear twice in the file list
3578
3579     F=$WDIR/$P
3580
3581     print "<p>""
3582     # If there's a diffs file, make diffs links
3583
3584     if [[ -f $F.cdiff.html ]]; then
3585         cdiff_url="$(print $P.cdiff.html | url_encode)"
3586         udiff_url="$(print $P.udiff.html | url_encode)"
3587         print "<a href=\"$cdiff_url\">Cdiffs</a>""
3588         print "<a href=\"$udiff_url\">Udiffs</a>""
3589
3590         if [[ -f $F.wdiff.html && -x $WDIFF ]]; then
3591             wdiff_url="$(print $P.wdiff.html | url_encode)"
3592             print "<a href=\"$wdiff_url\">Wdiffs</a>""
3593         fi
3594
3595         sdiff_url="$(print $P.sdiff.html | url_encode)"
3596         print "<a href=\"$sdiff_url\">Sdiffs</a>""
3597
3598         frames_url="$(print $P.frames.html | url_encode)"
3599         print "<a href=\"$frames_url\">Frames</a>""
3600
3601     else
3602         print " ----- ----- -----"
3603
3604         if [[ -x $WDIFF ]]; then
3605             print " -----"
3606         fi
3607
3608         print " -----"
3609     fi
3610
3611     # If there's an old file, make the link
3612
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
3619
3620     # If there's an new file, make the link
3621
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 $F.patch ]]; then
3630         patch_url=$(printf $P.patch | url_encode)"
3631         print "<a href=\"$patch_url\">Patch</a>"
3632     else
3633         print " ----"
3634     fi
3635
3636     if [[ -f $WDIR/raw_files/new/$P ]]; then
3637         rawfiles_url=$(printf raw_files/new/$P | url_encode)"
3638         print "<a href=\"$rawfiles_url\">Raw</a>"
3639     else
3640         print " ---"
3641     fi
3642
3643     print "<b>$P</b>"
3644
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
3659
3660     # If there's an old file, but no new file, the file was deleted
3661     if [[ -f $F.html && ! -f $F.html ]]; then
3662         print "<i>(deleted)</i>"
3663     fi
3664
3665     #
3666     # Check for usr/closed and deleted_files/usr/closed
3667     #
3668     if [ ! -z "$Oflag" ]; then
3669         if [[ $P == usr/closed/* || \
3670               $P == deleted_files/usr/closed/* ]]; then
3671             print "&nbsp;&nbsp;<i>Closed source: omitted from \
3672                   this review</i>"
3673         fi
3674     fi
3675
3676     print "</p>"
3677     # Insert delta comments
3678
3679     print "<blockquote><pre>" 
3680     getComments html $P $PP
3681     print "</pre>" 
3682
3683     # Add additional comments comment
3684
3685     print "<!-- Add comments to explain changes in $P here -->" 
3686
3687     # Add count of changes.
3688
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
3697
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
3702
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
3707
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
3712
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
3728
3729     print "</blockquote>" 
3730 done
3731
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>" 
3742
3743 exec 1<&-                                # Close FD 1.
3744 exec 1<&3                                # dup FD 3 to restore stdout.
3745 exec 3<&-                                # close FD 3.
3746
3747 print "Done." 
3748
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
3756
3757 if [[ -n $Uflag ]]; then
3758     upload_webrev
3759     exit $?

```

