```
**********************************************************
   83819 Wed Jun 12 20:55:02 2013
new/usr/src/tools/scripts/webrev.sh
teamware must die
**********************************************************
______unchanged_portion_omitted_

 591 #
 592 # sdiff_to_html
 593 #
 594 # This function takes two files as arguments, obtains their diff, and
 595 # processes the diff output to present the files as an HTML document with
 596 # the files displayed side-by-side, differences shown in color.  It also
 597 # takes a delta comment, rendered as an HTML snippet, as the third
 598 # argument.  The function takes two files as arguments, then the name of
 599 # file, the path, and the comment.  The HTML will be delivered on stdout,
 600 # e.g.
 601 #
 602 #   $ sdiff_to_html old/usr/src/tools/scripts/webrev.sh \
 603 #         new/usr/src/tools/scripts/webrev.sh \
 604 #         webrev.sh usr/src/tools/scripts \
 605 #         '<a href="http://monaco.sfbay.sun.com/detail.jsp?cr=1234567">
 606 #          1234567</a> my bugid' > <file>.html
 607 #
 608 # framed_sdiff() is then called which creates $2.frames.html
 609 # in the webrev tree.
 610 #
 611 # FYI: This function is rather unusual in its use of awk.  The initial
 612 # diff run produces conventional diff output showing changed lines mixed
 613 # with editing codes.  The changed lines are ignored - we're interested in
 614 # the editing codes, e.g.
 615 #
 616 #      8c8
 617 #      57a61
 618 #      63c66,76
 619 #      68,93d80
 620 #      106d90
 621 #      108,110d91
 622 #
 623 #  These editing codes are parsed by the awk script and used to generate
 624 #  another awk script that generates HTML, e.g the above lines would turn
 625 #  into something like this:
 626 #
 627 #      BEGIN { printf "<pre>\n" }
 628 #      function sp(n) {for (i=0;i<n;i++)printf "\n"}
 629 #      function wl(n) {printf "<font color=%s>%4d %s </font>\n", n, NR, $0}
 630 #      NR==8          {wl("#7A7ADD");next}
 631 #      NR==54         {wl("#7A7ADD");sp(3);next}
 632 #      NR==56         {wl("#7A7ADD");next}
 633 #      NR==57         {wl("black");printf "\n"; next}
 634 #        :              :
 635 #
 636 #  This script is then run on the original source file to generate the
 637 #  HTML that corresponds to the source file.
 638 #
 639 #  The two HTML files are then combined into a single piece of HTML that
 640 #  uses an HTML table construct to present the files side by side.  You'll
 641 #  notice that the changes are color-coded:
 642 #
 643 #   black     - unchanged lines
 644 #   blue      - changed lines
 645 #   bold blue - new lines
 646 #   brown     - deleted lines
 647 #
 648 #  Blank lines are inserted in each file to keep unchanged lines in sync
 649 #  (side-by-side).  This format is familiar to users of sdiff(1).
```



```
 649 #  (side-by-side).  This format is familiar to users of sdiff(1) or
 650 #  Teamware's filemerge tool.
 650 #
 651 sdiff_to_html()
 652 {
 653         diff -b $1 $2 > /tmp/$$.diffs

 655         TNAME=$3
 656         TPATH=$4
 657         COMMENT=$5

 659         #
 660         #  Now we have the diffs, generate the HTML for the old file.
 661         #
 662         $AWK '
 663         BEGIN    {
 664                 printf "function sp(n) {for (i=0;i<n;i++)printf \"\\n\"}\n"
 665                 printf "function removed() "
 666                 printf "{printf \"<span class=\\\"removed\\\">%%4d %%s</span>\\n
 667                 printf "function changed() "
 668                 printf "{printf \"<span class=\\\"changed\\\">%%4d %%s</span>\\n
 669                 printf "function bl() {printf \"%%4d %%s\\n\", NR, $0}\n"
 670 }
______unchanged_portion_omitted_


 925 #
 926 # fix_postscript
 927 #

1433 #
1433 # comments_from_teamware {text|html} parent-file child-file
1434 #
1435 # Find the first delta in the child that's not in the parent.  Get the
1436 # newest delta from the parent, get all deltas from the child starting
1437 # with that delta, and then get all info starting with the second oldest
1438 # delta in that list (the first delta unique to the child).
1439 #
1440 # This code adapted from Bill Shannon's "spc" script
1441 #
1442 comments_from_teamware()
1443 {
1444         fmt=$1
1445         pfile=$PWS/$2
1446         cfile=$CWS/$3

1448         if [[ ! -f $PWS/${2%/*}/SCCS/s.${2##*/} && -n $RWS ]]; then
1449                 pfile=$RWS/$2
1450         fi

1452         if [[ -f $pfile ]]; then
1453                 psid=$($SCCS prs -d:I: $pfile 2>/dev/null)
1454         else
1455                 psid=1.1
1456         fi

1458         set -A sids $($SCCS prs -l -r$psid -d:I: $cfile 2>/dev/null)
1459         N=${#sids[@]}

1461         nawkprg='
1462                 /^COMMENTS:/    {p=1; continue}
1463                 /^D [0-9]+\.[0-9]+/ {printf "--- %s ---\n", $2; p=0; }
1464                 NF == 0u        { continue }
1465                 {if (p==0) continue; print $0 }'

1467         if [[ $N -ge 2 ]]; then
```

```
                sid1=${sids[$((N-2))]}  # Gets 2nd to last sid

                if [[ $fmt == "text" ]]; then
                        $SCCS prs -l -r$sid1 $cfile  2>/dev/null | \
                            $AWK "$nawkprg"
                        return
                fi

                $SCCS prs -l -r$sid1 $cfile  2>/dev/null | \
                    html_quote | its2url | $AWK "$nawkprg"
        fi
}

#
# comments_from_wx {text|html} filepath
#
# Given the pathname of a file, find its location in a "wx" active
# file list and print the following comment.  Output is either text or
# HTML; if the latter, embedded bugids (sequence of 5 or more digits)
# are turned into URLs.
#
# This is also used with Mercurial and the file list provided by hg-active.
#
comments_from_wx()
{
        typeset fmt=$1
        typeset p=$2

        comm='$AWK '
        $1 == "'$p'" {
                do getline ; while (NF > 0)
                getline
                while (NF > 0) { print ; getline }
                exit
        }' < $wxfile'

        if [[ -z $comm ]]; then
                comm="*** NO COMMENTS ***"
        fi

        if [[ $fmt == "text" ]]; then
                print -- "$comm"
                return
        fi

        print -- "$comm" | html_quote | its2url

}

#
# getcomments {text|html} filepath parentpath
#
# Fetch the comments depending on what SCM mode we're in.
#
getcomments()
{
        typeset fmt=$1
        typeset p=$2
        typeset pp=$3

        if [[ -n $Nflag ]]; then
                return
        fi
        #
        # Mercurial support uses a file list in wx format, so this
        # will be used there, too
```



```
        #
        if [[ -n $wxfile ]]; then
                comments_from_wx $fmt $p
        else
                if [[ $SCM_MODE == "teamware" ]]; then
                        comments_from_teamware $fmt $pp $p
                fi
        fi
}
```

_____**unchanged_portion_omitted_**

```
#
# flist_from_teamware [ <args-to-putback-n> ]
#
# Generate the file list by extracting file names from a putback -n.  Some
# names may come from the "update/create" messages and others from the
# "currently checked out" warning.  Renames are detected here too.  Extract
# values for CODEMGR_WS and CODEMGR_PARENT from the output of the putback
# -n as well, but remove them if they are already defined.
#
function flist_from_teamware
{
        if [[ -n $codemgr_parent && -z $parent_webrev ]]; then
                if [[ ! -d $codemgr_parent/Codemgr_wsdata ]]; then
                        print -u2 "parent $codemgr_parent doesn't look like a" \
                            "valid teamware workspace"
                        exit 1
                fi
                parent_args="-p $codemgr_parent"
        fi

        print " File list from: 'putback -n $parent_args $*' ... \c"

        putback -n $parent_args $* 2>&1 |
            $AWK '
                /^update:|^create:/     {print $2}
                /^Parent workspace:/    {printf("CODEMGR_PARENT=%s\n",$3)}
                /^Child workspace:/     {printf("CODEMGR_WS=%s\n",$3)}
                /^The following files are currently checked out/ {p = 1; continu
                NF == 0                 {p=0 ; continue}
                /^rename/               {old=$3}
                $1 == "to:"             {print $2, old}
                /^"/                    {continue}
                p == 1                  {print $1}' |
            sort -r -k 1,1 -u | sort > $FLIST

        print " Done."
}

#
# Call hg-active to get the active list output in the wx active list format
#
function hg_active_wxfile
{
        typeset child=$1
        typeset parent=$2

        TMPFLIST=/tmp/$$.active
        $HG_ACTIVE -w $child -p $parent -o $TMPFLIST
        wxfile=$TMPFLIST
}
```

_____**unchanged_portion_omitted_**

```
function look_for_prog
{
        typeset path
```

```
1822         typeset ppath
1823         typeset progname=$1

1825         ppath=$PATH
1826         ppath=$ppath:/usr/sfw/bin:/usr/bin:/usr/sbin
1827         ppath=$ppath:/opt/onbld/bin:/opt/onbld/bin/‘uname -p‘
1917         ppath=$ppath:/opt/teamware/bin:/opt/onbld/bin
1918         ppath=$ppath:/opt/onbld/bin/‘uname -p‘

1829         PATH=$ppath prog=‘whence $progname‘
1830         if [[ -n $prog ]]; then
1831                 print $prog
1832         fi
1833 }
```

_____***unchanged_portion_omitted_***

```
1939 function build_old_new_teamware
1940 {
1941         typeset olddir="$1"
1942         typeset newdir="$2"

1944         # If the child’s version doesn’t exist then
1945         # get a readonly copy.

1947         if [[ ! -f $CWS/$DIR/$F && -f $CWS/$DIR/SCCS/s.$F ]]; then
1948                 $SCCS get -s -p $CWS/$DIR/$F > $CWS/$DIR/$F
1949         fi

1951         # The following two sections propagate file permissions the
1952         # same way SCCS does.  If the file is already under version
1953         # control, always use permissions from the SCCS/s.file.  If
1954         # the file is not under SCCS control, use permissions from the
1955         # working copy.  In all cases, the file copied to the webrev
1956         # is set to read only, and group/other permissions are set to
1957         # match those of the file owner.  This way, even if the file
1958         # is currently checked out, the webrev will display the final
1959         # permissions that would result after check in.

1961         #
1962         # Snag new version of file.
1963         #
1964         rm -f $newdir/$DIR/$F
1965         cp $CWS/$DIR/$F $newdir/$DIR/$F
1966         if [[ -f $CWS/$DIR/SCCS/s.$F ]]; then
1967                 chmod ‘get_file_mode $CWS/$DIR/SCCS/s.$F‘ \
1968                     $newdir/$DIR/$F
1969         fi
1970         chmod u-w,go=u $newdir/$DIR/$F

1972         #
1973         # Get the parent’s version of the file. First see whether the
1974         # child’s version is checked out and get the parent’s version
1975         # with keywords expanded or unexpanded as appropriate.
1976         #
1977         if [[ -f $PWS/$PDIR/$PF && ! -f $PWS/$PDIR/SCCS/s.$PF && \
1978             ! -f $PWS/$PDIR/SCCS/p.$PF ]]; then
1979                 # Parent is not a real workspace, but just a raw
1980                 # directory tree - use the file that’s there as
1981                 # the old file.

1983                 rm -f $olddir/$PDIR/$PF
1984                 cp $PWS/$PDIR/$PF $olddir/$PDIR/$PF
1985         else
1986                 if [[ -f $PWS/$PDIR/SCCS/s.$PF ]]; then
1987                         real_parent=$PWS
1988                 else
```



```
1989                         real_parent=$RWS
1990                 fi

1992                 rm -f $olddir/$PDIR/$PF

1994                 if [[ -f $real_parent/$PDIR/$PF ]]; then
1995                         if [ -f $CWS/$DIR/SCCS/p.$F ]; then
1996                                 $SCCS get -s -p -k $real_parent/$PDIR/$PF > \
1997                                     $olddir/$PDIR/$PF
1998                         else
1999                                 $SCCS get -s -p    $real_parent/$PDIR/$PF > \
2000                                     $olddir/$PDIR/$PF
2001                         fi
2002                         chmod ‘get_file_mode $real_parent/$PDIR/SCCS/s.$PF‘ \
2003                             $olddir/$PDIR/$PF
2004                 fi
2005         fi
2006         if [[ -f $olddir/$PDIR/$PF ]]; then
2007                 chmod u-w,go=u $olddir/$PDIR/$PF
2008         fi
2009 }
```

```
1848 function build_old_new_mercurial
1849 {
1850         typeset olddir="$1"
1851         typeset newdir="$2"
1852         typeset old_mode=
1853         typeset new_mode=
1854         typeset file

1856         #
1857         # Get old file mode, from the parent revision manifest entry.
1858         # Mercurial only stores a "file is executable" flag, but the
1859         # manifest will display an octal mode "644" or "755".
1860         #
1861         if [[ "$PDIR" == "." ]]; then
1862                 file="$PF"
1863         else
1864                 file="$PDIR/$PF"
1865         fi
1866         file=‘echo $file | $SED ’s#/#\\\/#g’‘
1867         # match the exact filename, and return only the permission digits
1868         old_mode=‘$SED -n -e "/^\\(...\\) . ${file}$/s//\\1/p" \
1869             < $HG_PARENT_MANIFEST‘

1871         #
1872         # Get new file mode, directly from the filesystem.
1873         # Normalize the mode to match Mercurial’s behavior.
1874         #
1875         new_mode=‘get_file_mode $CWS/$DIR/$F‘
1876         if [[ -n "$new_mode" ]]; then
1877                 if [[ "$new_mode" = *[1357]* ]]; then
1878                         new_mode=755
1879                 else
1880                         new_mode=644
1881                 fi
1882         fi

1884         #
1885         # new version of the file.
1886         #
1887         rm -rf $newdir/$DIR/$F
1888         if [[ -e $CWS/$DIR/$F ]]; then
1889                 cp $CWS/$DIR/$F $newdir/$DIR/$F
1890                 if [[ -n $new_mode ]]; then
1891                         chmod $new_mode $newdir/$DIR/$F
```

```
1892                 else
1893                         # should never happen
1894                         print -u2 "ERROR: set mode of $newdir/$DIR/$F"
1895                 fi
1896         fi

1898         #
1899         # parent's version of the file
1900         #
1901         # Note that we get this from the last version common to both
1902         # ourselves and the parent.  References are via $CWS since we have no
1903         # guarantee that the parent workspace is reachable via the filesystem.
1904         #
1905         if [[ -n $parent_webrev && -e $PWS/$PDIR/$PF ]]; then
1906                 cp $PWS/$PDIR/$PF $olddir/$PDIR/$PF
1907         elif [[ -n $HG_PARENT ]]; then
1908                 hg cat -R $CWS -r $HG_PARENT $CWS/$PDIR/$PF > \
1909                     $olddir/$PDIR/$PF 2>/dev/null

1911                 if (( $? != 0 )); then
1912                         rm -f $olddir/$PDIR/$PF
1913                 else
1914                         if [[ -n $old_mode ]]; then
1915                                 chmod $old_mode $olddir/$PDIR/$PF
1916                         else
1917                                 # should never happen
1918                                 print -u2 "ERROR: set mode of $olddir/$PDIR/$PF"
1919                         fi
1920                 fi
1921         fi
1922 }
```

*_____unchanged_portion_omitted_*

```
2022 function build_old_new
2023 {
2024         typeset WDIR=$1
2025         typeset PWS=$2
2026         typeset PDIR=$3
2027         typeset PF=$4
2028         typeset CWS=$5
2029         typeset DIR=$6
2030         typeset F=$7

2032         typeset olddir="$WDIR/raw_files/old"
2033         typeset newdir="$WDIR/raw_files/new"

2035         mkdir -p $olddir/$PDIR
2036         mkdir -p $newdir/$DIR

2038         if [[ $SCM_MODE == "mercurial" ]]; then
2201         if [[ $SCM_MODE == "teamware" ]]; then
2202                 build_old_new_teamware "$olddir" "$newdir"
2203         elif [[ $SCM_MODE == "mercurial" ]]; then
2039                 build_old_new_mercurial "$olddir" "$newdir"
2040         elif [[ $SCM_MODE == "git" ]]; then
2041                 build_old_new_git "$olddir" "$newdir"
2042         elif [[ $SCM_MODE == "subversion" ]]; then
2043                 build_old_new_subversion "$olddir" "$newdir"
2044         elif [[ $SCM_MODE == "unknown" ]]; then
2045                 build_old_new_unknown "$olddir" "$newdir"
2046         fi

2048         if [[ ! -f $olddir/$PDIR/$PF && ! -f $newdir/$DIR/$F ]]; then
2049                 print "*** Error: file not in parent or child"
2050                 return 1
2051         fi
```



```
2052         return 0
2053 }


2056 #
2057 # Usage message.
2058 #
2059 function usage
2060 {
2061         print 'Usage:\twebrev [common-options]
2062         webrev [common-options] ( <file> | - )
2063         webrev [common-options] -w <wx file>

2065 Options:
2066         -C <filename>: Use <filename> for the information tracking configuration
2067         -D: delete remote webrev
2068         -i <filename>: Include <filename> in the index.html file.
2069         -I <filename>: Use <filename> for the information tracking registry.
2070         -n: do not generate the webrev (useful with -U)
2071         -O: Print bugids/arc cases suitable for OpenSolaris.
2072         -o <outdir>: Output webrev to specified directory.
2073         -p <compare-against>: Use specified parent wkspc or basis for comparison
2074         -t <remote_target>: Specify remote destination for webrev upload
2075         -U: upload the webrev to remote destination
2076         -w <wxfile>: Use specified wx active file.

2078 Environment:
2079         WDIR: Control the output directory.
2080         WEBREV_TRASH_DIR: Set directory for webrev delete.

2247 SCM Specific Options:
2248         TeamWare: webrev [common-options] -l [arguments to 'putback']

2082 SCM Environment:
2083         CODEMGR_WS: Workspace location.
2084         CODEMGR_PARENT: Parent workspace location.
2085 '

2086         exit 2
2087 }

2089 #
2090 #
2091 # Main program starts here
2092 #
2093 #

2095 trap "rm -f /tmp/$$.* ; exit" 0 1 2 3 15

2097 set +o noclobber

2099 PATH=$(/bin/dirname "$(whence $0)"):$PATH

2101 [[ -z $WDIFF ]] && WDIFF=`look_for_prog wdiff`
2102 [[ -z $WX ]] && WX=`look_for_prog wx`
2103 [[ -z $HG_ACTIVE ]] && HG_ACTIVE=`look_for_prog hg-active`
2104 [[ -z $GIT ]] && GIT=`look_for_prog git`
2105 [[ -z $WHICH_SCM ]] && WHICH_SCM=`look_for_prog which_scm`
2106 [[ -z $CODEREVIEW ]] && CODEREVIEW=`look_for_prog codereview`
2107 [[ -z $PS2PDF ]] && PS2PDF=`look_for_prog ps2pdf`
2108 [[ -z $PERL ]] && PERL=`look_for_prog perl`
2109 [[ -z $RSYNC ]] && RSYNC=`look_for_prog rsync`
2110 [[ -z $SCCS ]] && SCCS=`look_for_prog sccs`
2111 [[ -z $AWK ]] && AWK=`look_for_prog nawk`
2112 [[ -z $AWK ]] && AWK=`look_for_prog gawk`
2113 [[ -z $AWK ]] && AWK=`look_for_prog awk`
```

```
2114 [[ -z $SCP ]] && SCP=‘look_for_prog scp‘
2115 [[ -z $SED ]] && SED=‘look_for_prog sed‘
2116 [[ -z $SFTP ]] && SFTP=‘look_for_prog sftp‘
2117 [[ -z $SORT ]] && SORT=‘look_for_prog sort‘
2118 [[ -z $MKTEMP ]] && MKTEMP=‘look_for_prog mktemp‘
2119 [[ -z $GREP ]] && GREP=‘look_for_prog grep‘
2120 [[ -z $FIND ]] && FIND=‘look_for_prog find‘

2122 # set name of trash directory for remote webrev deletion
2123 TRASH_DIR=".trash"
2124 [[ -n $WEBREV_TRASH_DIR ]] && TRASH_DIR=$WEBREV_TRASH_DIR

2126 if [[ ! -x $PERL ]]; then
2127         print -u2 "Error: No perl interpreter found.  Exiting."
2128         exit 1
2129 fi

2131 if [[ ! -x $WHICH_SCM ]]; then
2132         print -u2 "Error: Could not find which_scm.  Exiting."
2133         exit 1
2134 fi

2136 #
2137 # These aren’t fatal, but we want to note them to the user.
2138 # We don’t warn on the absence of ’wx’ until later when we’ve
2139 # determined that we actually need to try to invoke it.
2140 #
2141 [[ ! -x $CODEREVIEW ]] && print -u2 "WARNING: codereview(1) not found."
2142 [[ ! -x $PS2PDF ]] && print -u2 "WARNING: ps2pdf(1) not found."
2143 [[ ! -x $WDIFF ]] && print -u2 "WARNING: wdiff not found."

2145 # Declare global total counters.
2146 integer TOTL TINS TDEL TMOD TUNC

2148 # default remote host for upload/delete
2149 typeset -r DEFAULT_REMOTE_HOST="cr.opensolaris.org"
2150 # prefixes for upload targets
2151 typeset -r rsync_prefix="rsync://"
2152 typeset -r ssh_prefix="ssh://"

2154 Cflag=
2155 Dflag=
2156 flist_mode=
2157 flist_file=
2158 iflag=
2159 Iflag=
2160 lflag=
2161 Nflag=
2162 nflag=
2163 Oflag=
2164 oflag=
2165 pflag=
2166 tflag=
2167 uflag=
2168 Uflag=
2169 wflag=
2170 remote_target=

2172 #
2173 # NOTE: when adding/removing options it is necessary to sync the list
2174 #       with usr/src/tools/onbld/hgext/cdm.py
2175 #
2176 while getopts "C:Di:I:lnNo:Op:t:Uw" opt
2177 do
2178         case $opt in
2179         C)      Cflag=1
```



```
2180                 ITSCONF=$OPTARG;;

2182         D)      Dflag=1;;

2184         i)      iflag=1
2185                 INCLUDE_FILE=$OPTARG;;

2187         I)      Iflag=1
2188                 ITSREG=$OPTARG;;

2190         #
2191         # If -l has been specified, we need to abort further options
2192         # processing, because subsequent arguments are going to be
2193         # arguments to ’putback -n’.
2194         #
2195         l)      lflag=1
2196                 break;;

2198         N)      Nflag=1;;

2200         n)      nflag=1;;

2202         O)      Oflag=1;;

2204         o)      oflag=1
2205                 # Strip the trailing slash to correctly form remote target.
2206                 WDIR=${OPTARG%/};;

2208         p)      pflag=1
2209                 codemgr_parent=$OPTARG;;

2211         t)      tflag=1
2212                 remote_target=$OPTARG;;

2214         U)      Uflag=1;;

2216         w)      wflag=1;;

2218         ?)      usage;;
2219         esac
2220 done

2222 FLIST=/tmp/$$.flist

2224 if [[ -n $wflag && -n $lflag ]]; then
2225         usage
2226 fi

2228 # more sanity checking
2229 if [[ -n $nflag && -z $Uflag ]]; then
2230         print "it does not make sense to skip webrev generation" \
2231             "without -U"
2232         exit 1
2233 fi

2235 if [[ -n $tflag && -z $Uflag && -z $Dflag ]]; then
2236         echo "remote target has to be used only for upload or delete"
2237         exit 1
2238 fi

2240 #
2241 # For the invocation "webrev -n -U" with no other options, webrev will assume
2242 # that the webrev exists in ${CWS}/webrev, but will upload it using the name
2243 # $(basename ${CWS}).  So we need to get CWS set before we skip any remaining
2244 # logic.
2245 #
```

```
2246 $WHICH_SCM | read SCM_MODE junk || exit 1
2247 if [[ $SCM_MODE == "mercurial" ]]; then
2416 if [[ $SCM_MODE == "teamware" ]]; then
2417         #
2418         # Teamware priorities:
2419         # 1. CODEMGR_WS from the environment
2420         # 2. workspace name
2421         #
2422         [[ -z $codemgr_ws && -n $CODEMGR_WS ]] && codemgr_ws=$CODEMGR_WS
2423         if [[ -n $codemgr_ws && ! -d $codemgr_ws ]]; then
2424                 print -u2 "$codemgr_ws: no such workspace"
2425                 exit 1
2426         fi
2427         [[ -z $codemgr_ws ]] && codemgr_ws=$(workspace name)
2428         codemgr_ws=$(cd $codemgr_ws;print $PWD)
2429         CODEMGR_WS=$codemgr_ws
2430         CWS=$codemgr_ws
2431 elif [[ $SCM_MODE == "mercurial" ]]; then
2248         #
2249         # Mercurial priorities:
2250         # 1. hg root from CODEMGR_WS environment variable
2251         # 1a. hg root from CODEMGR_WS/usr/closed if we're somewhere under
2252         #    usr/closed when we run webrev
2253         # 2. hg root from directory of invocation
2254         #
2255         if [[ ${PWD} =~ "usr/closed" ]]; then
2256                 testparent=${CODEMGR_WS}/usr/closed
2257                 # If we're in OpenSolaris mode, we enforce a minor policy:
2258                 # help to make sure the reviewer doesn't accidentally publish
2259                 # source which is under usr/closed
2260                 if [[ -n "$Oflag" ]]; then
2261                         print -u2 "OpenSolaris output not permitted with" \
2262                             "usr/closed changes"
2263                         exit 1
2264                 fi
2265         else
2266                 testparent=${CODEMGR_WS}
2267         fi
2268         [[ -z $codemgr_ws && -n $testparent ]] && \
2269             codemgr_ws=$(hg root -R $testparent 2>/dev/null)
2270         [[ -z $codemgr_ws ]] && codemgr_ws=$(hg root 2>/dev/null)
2271         CWS=$codemgr_ws
2272 elif [[ $SCM_MODE == "git" ]]; then
2273         #
2274         # Git priorities:
2275         # 1. git rev-parse --git-dir from CODEMGR_WS environment variable
2276         # 2. git rev-parse --git-dir from directory of invocation
2277         #
2278         [[ -z $codemgr_ws && -n $CODEMGR_WS ]] && \
2279             codemgr_ws=$($GIT --git-dir=$CODEMGR_WS/.git rev-parse --git-dir \
2280                 2>/dev/null)
2281         [[ -z $codemgr_ws ]] && \
2282             codemgr_ws=$($GIT rev-parse --git-dir 2>/dev/null)

2284         if [[ "$codemgr_ws" == ".git" ]]; then
2285                 codemgr_ws="${PWD}/${codemgr_ws}"
2286         fi

2288         codemgr_ws=$(dirname $codemgr_ws) # Lose the '/.git'
2289         CWS="$codemgr_ws"
2290 elif [[ $SCM_MODE == "subversion" ]]; then
2291         #
2292         # Subversion priorities:
2293         # 1. CODEMGR_WS from environment
2294         # 2. Relative path from current directory to SVN repository root
2295         #
```



```
2296         if [[ -n $CODEMGR_WS && -d $CODEMGR_WS/.svn ]]; then
2297                 CWS=$CODEMGR_WS
2298         else
2299                 svn info | while read line; do
2300                         if [[ $line == "URL: "* ]]; then
2301                                 url=${line#URL: }
2302                         elif [[ $line == "Repository Root: "* ]]; then
2303                                 repo=${line#Repository Root: }
2304                         fi
2305                 done

2307                 rel=${url#$repo}
2308                 CWS=${PWD%$rel}
2309         fi
2310 fi

2312 #
2313 # If no SCM has been determined, take either the environment setting
2314 # setting for CODEMGR_WS, or the current directory if that wasn't set.
2315 #
2316 if [[ -z ${CWS} ]]; then
2317         CWS=${CODEMGR_WS:-.}
2318 fi

2320 #
2321 # If the command line options indicate no webrev generation, either
2322 # explicitly (-n) or implicitly (-D but not -U), then there's a whole
2323 # ton of logic we can skip.
2324 #
2325 # Instead of increasing indentation, we intentionally leave this loop
2326 # body open here, and exit via break from multiple points within.
2327 # Search for DO_EVERYTHING below to find the break points and closure.
2328 #
2329 for do_everything in 1; do

2331 # DO_EVERYTHING: break point
2332 if [[ -n $nflag || ( -z $Uflag && -n $Dflag ) ]]; then
2333         break
2334 fi

2336 #
2337 # If this manually set as the parent, and it appears to be an earlier webrev,
2338 # then note that fact and set the parent to the raw_files/new subdirectory.
2339 #
2340 if [[ -n $pflag && -d $codemgr_parent/raw_files/new ]]; then
2341         parent_webrev=$(readlink -f "$codemgr_parent")
2342         codemgr_parent=$(readlink -f "$codemgr_parent/raw_files/new")
2343 fi

2345 if [[ -z $wflag && -z $lflag ]]; then
2346         shift $(($OPTIND - 1))

2348         if [[ $1 == "-" ]]; then
2349                 cat > $FLIST
2350                 flist_mode="stdin"
2351                 flist_done=1
2352                 shift
2353         elif [[ -n $1 ]]; then
2354                 if [[ ! -r $1 ]]; then
2355                         print -u2 "$1: no such file or not readable"
2356                         usage
2357                 fi
2358                 cat $1 > $FLIST
2359                 flist_mode="file"
2360                 flist_file=$1
2361                 flist_done=1
```

```
2362                 shift
2363         else
2364                 flist_mode="auto"
2365         fi
2366 fi

2368 #
2369 # Before we go on to further consider -l and -w, work out which SCM we think
2370 # is in use.
2371 #
2372 case "$SCM_MODE" in
2373 mercurial|git|subversion)
2557 teamware|mercurial|git|subversion)
2374         ;;
2375 unknown)
2376         if [[ $flist_mode == "auto" ]]; then
2377                 print -u2 "Unable to determine SCM in use and file list not spec
2378                 print -u2 "See which_scm(1) for SCM detection information."
2379                 exit 1
2380         fi
2381         ;;
2382 *)
2383         if [[ $flist_mode == "auto" ]]; then
2384                 print -u2 "Unsupported SCM in use ($SCM_MODE) and file list not
2385                 exit 1
2386         fi
2387         ;;
2388 esac

2390 print -u2 "   SCM detected: $SCM_MODE"

2392 if [[ -n $wflag ]]; then
2576 if [[ -n $lflag ]]; then
2577         #
2578         # If the -l flag is given instead of the name of a file list,
2579         # then generate the file list by extracting file names from a
2580         # putback -n.
2581         #
2582         shift $(($OPTIND - 1))
2583         if [[ $SCM_MODE == "teamware" ]]; then
2584                 flist_from_teamware "$*"
2585         else
2586                 print -u2 -- "Error: -l option only applies to TeamWare"
2587                 exit 1
2588         fi
2589         flist_done=1
2590         shift $#
2591 elif [[ -n $wflag ]]; then
2393         #
2394         # If the -w is given then assume the file list is in Bonwick's "wx"
2395         # command format, i.e.  pathname lines alternating with SCCS comment
2396         # lines with blank lines as separators.  Use the SCCS comments later
2397         # in building the index.html file.
2398         #
2399         shift $(($OPTIND - 1))
2400         wxfile=$1
2401         if [[ -z $wxfile && -n $CODEMGR_WS ]]; then
2402                 if [[ -r $CODEMGR_WS/wx/active ]]; then
2403                         wxfile=$CODEMGR_WS/wx/active
2404                 fi
2405         fi

2407         [[ -z $wxfile ]] && print -u2 "wx file not specified, and could not " \
2408             "be auto-detected (check \$CODEMGR_WS)" && exit 1

2410         if [[ ! -r $wxfile ]]; then
```



```
2411                 print -u2 "$wxfile: no such file or not readable"
2412                 usage
2413         fi

2415         print -u2 " File list from: wx 'active' file '$wxfile' ... \c"
2416         flist_from_wx $wxfile
2417         flist_done=1
2418         if [[ -n "$*" ]]; then
2419                 shift
2420         fi
2421 elif [[ $flist_mode == "stdin" ]]; then
2422         print -u2 " File list from: standard input"
2423 elif [[ $flist_mode == "file" ]]; then
2424         print -u2 " File list from: $flist_file"
2425 fi

2427 if [[ $# -gt 0 ]]; then
2428         print -u2 "WARNING: unused arguments: $*"
2429 fi

2431 #
2432 # Before we entered the DO_EVERYTHING loop, we should have already set CWS
2433 # and CODEMGR_WS as needed.  Here, we set the parent workspace.
2434 #

2436 if [[ $SCM_MODE == "mercurial" ]]; then
2635 if [[ $SCM_MODE == "teamware" ]]; then

2637         #
2638         # Teamware priorities:
2639         #
2640         #       1) via -p command line option
2641         #       2) in the user environment
2642         #       3) in the flist
2643         #       4) automatically based on the workspace
2644         #

2646         #
2647         # For 1, codemgr_parent will already be set.  Here's 2:
2648         #
2649         [[ -z $codemgr_parent && -n $CODEMGR_PARENT ]] && \
2650             codemgr_parent=$CODEMGR_PARENT
2651         if [[ -n $codemgr_parent && ! -d $codemgr_parent ]]; then
2652                 print -u2 "$codemgr_parent: no such directory"
2653                 exit 1
2654         fi

2656         #
2657         # If we're in auto-detect mode and we haven't already gotten the file
2658         # list, then see if we can get it by probing for wx.
2659         #
2660         if [[ -z $flist_done && $flist_mode == "auto" && -n $codemgr_ws ]]; then
2661                 if [[ ! -x $WX ]]; then
2662                         print -u2 "WARNING: wx not found!"
2663                 fi

2665                 #
2666                 # We need to use wx list -w so that we get renamed files, etc.
2667                 # but only if a wx active file exists-- otherwise wx will
2668                 # hang asking us to initialize our wx information.
2669                 #
2670                 if [[ -x $WX && -f $codemgr_ws/wx/active ]]; then
2671                         print -u2 " File list from: 'wx list -w' ... \c"
2672                         $WX list -w > $FLIST
2673                         $WX comments > /tmp/$$.wx_comments
2674                         wxfile=/tmp/$$.wx_comments
```

```
2675                        print -u2 "done"
2676                        flist_done=1
2677                fi
2678        fi

2680        #
2681        # If by hook or by crook we've gotten a file list by now (perhaps
2682        # from the command line), eval it to extract environment variables from
2683        # it: This is method 3 for finding the parent.
2684        #
2685        if [[ -z $flist_done ]]; then
2686                flist_from_teamware
2687        fi
2688        env_from_flist

2690        #
2691        # (4) If we still don't have a value for codemgr_parent, get it
2692        # from workspace.
2693        #
2694        [[ -z $codemgr_parent ]] && codemgr_parent=`workspace parent`
2695        if [[ ! -d $codemgr_parent ]]; then
2696                print -u2 "$CODEMGR_PARENT: no such parent workspace"
2697                exit 1
2698        fi

2700        PWS=$codemgr_parent

2702        [[ -n $parent_webrev ]] && RWS=$(workspace parent $CWS)

2704 elif [[ $SCM_MODE == "mercurial" ]]; then
2437        #
2438        # Parent can either be specified with -p
2439        # Specified with CODEMGR_PARENT in the environment
2440        # or taken from hg's default path.
2441        #

2443        if [[ -z $codemgr_parent && -n $CODEMGR_PARENT ]]; then
2444                codemgr_parent=$CODEMGR_PARENT
2445        fi

2447        if [[ -z $codemgr_parent ]]; then
2448                codemgr_parent=`hg path -R $codemgr_ws default 2>/dev/null`
2449        fi

2451        PWS=$codemgr_parent

2453        #
2454        # If the parent is a webrev, we want to do some things against
2455        # the natural workspace parent (file list, comments, etc)
2456        #
2457        if [[ -n $parent_webrev ]]; then
2458                real_parent=$(hg path -R $codemgr_ws default 2>/dev/null)
2459        else
2460                real_parent=$PWS
2461        fi

2463        #
2464        # If hg-active exists, then we run it.  In the case of no explicit
2465        # flist given, we'll use it for our comments.  In the case of an
2466        # explicit flist given we'll try to use it for comments for any
2467        # files mentioned in the flist.
2468        #
2469        if [[ -z $flist_done ]]; then
2470                flist_from_mercurial $CWS $real_parent
2471                flist_done=1
2472        fi
```



```
2474        #
2475        # If we have a file list now, pull out any variables set
2476        # therein.  We do this now (rather than when we possibly use
2477        # hg-active to find comments) to avoid stomping specifications
2478        # in the user-specified flist.
2479        #
2480        if [[ -n $flist_done ]]; then
2481                env_from_flist
2482        fi

2484        #
2485        # Only call hg-active if we don't have a wx formatted file already
2486        #
2487        if [[ -x $HG_ACTIVE && -z $wxfile ]]; then
2488                print "  Comments from: hg-active -p $real_parent ...\c"
2489                hg_active_wxfile $CWS $real_parent
2490                print " Done."
2491        fi

2493        #
2494        # At this point we must have a wx flist either from hg-active,
2495        # or in general.  Use it to try and find our parent revision,
2496        # if we don't have one.
2497        #
2498        if [[ -z $HG_PARENT ]]; then
2499                eval `$SED -e "s/#.*$//" $wxfile | $GREP HG_PARENT=`
2500        fi

2502        #
2503        # If we still don't have a parent, we must have been given a
2504        # wx-style active list with no HG_PARENT specification, run
2505        # hg-active and pull an HG_PARENT out of it, ignore the rest.
2506        #
2507        if [[ -z $HG_PARENT && -x $HG_ACTIVE ]]; then
2508                $HG_ACTIVE -w $codemgr_ws -p $real_parent | \
2509                    eval `$SED -e "s/#.*$//" | $GREP HG_PARENT=`
2510        elif [[ -z $HG_PARENT ]]; then
2511                print -u2 "Error: Cannot discover parent revision"
2512                exit 1
2513        fi

2515        pnode=$(trim_digest $HG_PARENT)
2516        PRETTY_PWS="${PWS} (at ${pnode})"
2517        cnode=$(hg parent -R $codemgr_ws --template '{node|short}' \
2518            2>/dev/null)
2519        PRETTY_CWS="${CWS} (at ${cnode})"}
2520 elif [[ $SCM_MODE == "git" ]]; then
2521        #
2522        # Parent can either be specified with -p, or specified with
2523        # CODEMGR_PARENT in the environment.
2524        #

2526        if [[ -z $codemgr_parent && -n $CODEMGR_PARENT ]]; then
2527                codemgr_parent=$CODEMGR_PARENT
2528        fi

2530        # Try to figure out the parent based on the branch the current
2531        # branch is tracking, if we fail, use origin/master
2532        this_branch=$($GIT branch | nawk '$1 == "*" { print $2 }')
2533        par_branch="origin/master"

2535        # If we're not on a branch there's nothing we can do
2536        if [[ $this_branch != "(no branch)" ]]; then
2537                $GIT for-each-ref
2538                    --format='%(refname:short) %(upstream:short)' refs/heads/ |
```

```
2539                    while read local remote; do
2540                        [[ "$local" == "$this_branch" ]] && par_branch="$remote"
2541                    done
2542            fi

2544            if [[ -z $codemgr_parent ]]; then
2545                    codemgr_parent=$par_branch
2546            fi
2547            PWS=$codemgr_parent

2549            #
2550            # If the parent is a webrev, we want to do some things against
2551            # the natural workspace parent (file list, comments, etc)
2552            #
2553            if [[ -n $parent_webrev ]]; then
2554                    real_parent=$par_branch
2555            else
2556                    real_parent=$PWS
2557            fi

2559            if [[ -z $flist_done ]]; then
2560                    flist_from_git "$CWS" "$real_parent"
2561                    flist_done=1
2562            fi

2564            #
2565            # If we have a file list now, pull out any variables set
2566            # therein.
2567            #
2568            if [[ -n $flist_done ]]; then
2569                    env_from_flist
2570            fi

2572            #
2573            # If we don't have a wx-format file list, build one we can pull change
2574            # comments from.
2575            #
2576            if [[ -z $wxfile ]]; then
2577                    print "  Comments from: git...\c"
2578                    git_wxfile "$CWS" "$real_parent"
2579                    print " Done."
2580            fi

2582            if [[ -z $GIT_PARENT ]]; then
2583                    GIT_PARENT=$($GIT merge-base "$real_parent" HEAD)
2584            fi
2585            if [[ -z $GIT_PARENT ]]; then
2586                    print -u2 "Error: Cannot discover parent revision"
2587                    exit 1
2588            fi

2590            pnode=$(trim_digest $GIT_PARENT)

2592            if [[ $real_parent == */* ]]; then
2593                    origin=$(echo $real_parent | cut -d/ -f1)
2594                    origin=$($GIT remote -v | \
2595                        $AWK '$1 == "'$origin'" { print $2; exit }')
2596                    PRETTY_PWS="${PWS} (${origin} at ${pnode})"
2597            else
2598                    PRETTY_PWS="${PWS} (at ${pnode})"
2599            fi

2601            cnode=$($GIT --git-dir=${codemgr_ws}/.git rev-parse --short=12 HEAD \
2602                2>/dev/null)
2603            PRETTY_CWS="${CWS} (at ${cnode})"
2604    elif [[ $SCM_MODE == "subversion" ]]; then
```



```
2606            #
2607            # We only will have a real parent workspace in the case one
2608            # was specified (be it an older webrev, or another checkout).
2609            #
2610            [[ -n $codemgr_parent ]] && PWS=$codemgr_parent

2612            if [[ -z $flist_done && $flist_mode == "auto" ]]; then
2613                    flist_from_subversion $CWS $OLDPWD
2614            fi
2615    else
2616        if [[ $SCM_MODE == "unknown" ]]; then
2617            print -u2 "    Unknown type of SCM in use"
2618        else
2619            print -u2 "    Unsupported SCM in use: $SCM_MODE"
2620        fi

2622        env_from_flist

2624        if [[ -z $CODEMGR_WS ]]; then
2625            print -u2 "SCM not detected/supported and CODEMGR_WS not specified"
2626            exit 1
2627        fi

2629        if [[ -z $CODEMGR_PARENT ]]; then
2630            print -u2 "SCM not detected/supported and CODEMGR_PARENT not specified"
2631            exit 1
2632        fi

2634        CWS=$CODEMGR_WS
2635        PWS=$CODEMGR_PARENT
2636    fi

2638    #
2639    # If the user didn't specify a -i option, check to see if there is a
2640    # webrev-info file in the workspace directory.
2641    #
2642    if [[ -z $iflag && -r "$CWS/webrev-info" ]]; then
2643            iflag=1
2644            INCLUDE_FILE="$CWS/webrev-info"
2645    fi

2647    if [[ -n $iflag ]]; then
2648            if [[ ! -r $INCLUDE_FILE ]]; then
2649                    print -u2 "include file '$INCLUDE_FILE' does not exist or is" \
2650                        "not readable."
2651                    exit 1
2652            else
2653                    #
2654                    # $INCLUDE_FILE may be a relative path, and the script alters
2655                    # PWD, so we just stash a copy in /tmp.
2656                    #
2657                    cp $INCLUDE_FILE /tmp/$$.include
2658            fi
2659    fi

2661    # DO_EVERYTHING: break point
2662    if [[ -n $Nflag ]]; then
2663            break
2664    fi

2666    typeset -A itsinfo
2667    typeset -r its_sed_script=/tmp/$$.its_sed
2668    valid_prefixes=
2669    if [[ -z $nflag ]]; then
2670            DEFREGFILE="$(/bin/dirname "$(whence $0)")/../etc/its.reg"
```

```
2671         if [[ -n $Iflag ]]; then
2672                 REGFILE=$ITSREG
2673         elif [[ -r $HOME/.its.reg ]]; then
2674                 REGFILE=$HOME/.its.reg
2675         else
2676                 REGFILE=$DEFREGFILE
2677         fi
2678         if [[ ! -r $REGFILE ]]; then
2679                 print "ERROR: Unable to read database registry file $REGFILE"
2680                 exit 1
2681         elif [[ $REGFILE != $DEFREGFILE ]]; then
2682                 print "   its.reg from: $REGFILE"
2683         fi

2685         $SED -e '/^#/d' -e '/^[         ]*$/d' $REGFILE | while read LINE; do

2687                 name=${LINE%%=*}
2688                 value="${LINE#*=}"

2690                 if [[ $name == PREFIX ]]; then
2691                         p=${value}
2692                         valid_prefixes="${p} ${valid_prefixes}"
2693                 else
2694                         itsinfo["${p}_${name}"]="${value}"
2695                 fi
2696         done


2699         DEFCONFFILE="$(/bin/dirname "$(whence $0)")/../etc/its.conf"
2700         CONFFILES=$DEFCONFFILE
2701         if [[ -r $HOME/.its.conf ]]; then
2702                 CONFFILES="${CONFFILES} $HOME/.its.conf"
2703         fi
2704         if [[ -n $Cflag ]]; then
2705                 CONFFILES="${CONFFILES} ${ITSCONF}"
2706         fi
2707         its_domain=
2708         its_priority=
2709         for cf in ${CONFFILES}; do
2710                 if [[ ! -r $cf ]]; then
2711                         print "ERROR: Unable to read database configuration file
2712                         exit 1
2713                 elif [[ $cf != $DEFCONFFILE ]]; then
2714                         print "       its.conf: reading $cf"
2715                 fi
2716                 $SED -e '/^#/d' -e '/^[         ]*$/d' $cf | while read LINE; do
2717                     eval "${LINE}"
2718                 done
2719         done

2721         #
2722         # If an information tracking system is explicitly identified by prefix,
2723         # we want to disregard the specified priorities and resolve it according
2724         #
2725         # To that end, we'll build a sed script to do each valid prefix in turn.
2726         #
2727         for p in ${valid_prefixes}; do
2728                 #
2729                 # When an informational URL was provided, translate it to a
2730                 # hyperlink.  When omitted, simply use the prefix text.
2731                 #
2732                 if [[ -z ${itsinfo["${p}_INFO"]} ]]; then
2733                         itsinfo["${p}_INFO"]=${p}
2734                 else
2735                         itsinfo["${p}_INFO"]="<a href=\\\"${itsinfo["${p}_INFO"]
2736                 fi
```



```
2738                 #
2739                 # Assume that, for this invocation of webrev, all references
2740                 # to this information tracking system should resolve through
2741                 # the same URL.
2742                 #
2743                 # If the caller specified -O, then always use EXTERNAL_URL.
2744                 #
2745                 # Otherwise, look in the list of domains for a matching
2746                 # INTERNAL_URL.
2747                 #
2748                 [[ -z $Oflag ]] && for d in ${its_domain}; do
2749                         if [[ -n ${itsinfo["${p}_INTERNAL_URL_${d}"]} ]]; then
2750                                 itsinfo["${p}_URL"]="${itsinfo[${p}_INTERNAL_URL
2751                                 break
2752                         fi
2753                 done
2754                 if [[ -z ${itsinfo["${p}_URL"]} ]]; then
2755                         itsinfo["${p}_URL"]="${itsinfo[${p}_EXTERNAL_URL]}"
2756                 fi

2758                 #
2759                 # Turn the destination URL into a hyperlink
2760                 #
2761                 itsinfo["${p}_URL"]="<a href=\\\"${itsinfo[${p}_URL]}\\\">&</a>"

2763                 # The character class below contains a literal tab
2764                 print "/^${p}[:         ]/ {
2765                                 s;${itsinfo[${p}_REGEX]};${itsinfo[${p}_URL]};g
2766                                 s;^${p};${itsinfo[${p}_INFO]};
2767                         }" >> ${its_sed_script}
2768         done

2770         #
2771         # The previous loop took care of explicit specification.  Now use
2772         # the configured priorities to attempt implicit translations.
2773         #
2774         for p in ${its_priority}; do
2775                 print "/^${itsinfo[${p}_REGEX]}[        ]/ {
2776                                 s;^${itsinfo[${p}_REGEX]};${itsinfo[${p}_URL]};g
2777                         }" >> ${its_sed_script}
2778         done
2779 fi

2781 #
2782 # Search for DO_EVERYTHING above for matching "for" statement
2783 # and explanation of this terminator.
2784 #
2785 done

2787 #
2788 # Output directory.
2789 #
2790 WDIR=${WDIR:-$CWS/webrev}

2792 #
2793 # Name of the webrev, derived from the workspace name or output directory;
2794 # in the future this could potentially be an option.
2795 #
2796 if [[ -n $oflag ]]; then
2797         WNAME=${WDIR##*/}
2798 else
2799         WNAME=${CWS##*/}
2800 fi

2802 # Make sure remote target is well formed for remote upload/delete.
```

```
2803 if [[ -n $Dflag || -n $Uflag ]]; then
2804         #
2805         # If remote target is not specified, build it from scratch using
2806         # the default values.
2807         #
2808         if [[ -z $tflag ]]; then
2809                 remote_target=${DEFAULT_REMOTE_HOST}:${WNAME}
2810         else
2811                 #
2812                 # Check upload target prefix first.
2813                 #
2814                 if [[ "${remote_target}" != ${rsync_prefix}* &&
2815                     "${remote_target}" != ${ssh_prefix}* ]]; then
2816                         print "ERROR: invalid prefix of upload URI" \
2817                             "($remote_target)"
2818                         exit 1
2819                 fi
2820                 #
2821                 # If destination specification is not in the form of
2822                 # host_spec:remote_dir then assume it is just remote hostname
2823                 # and append a colon and destination directory formed from
2824                 # local webrev directory name.
2825                 #
2826                 typeset target_no_prefix=${remote_target##*://}
2827                 if [[ ${target_no_prefix} == *:* ]]; then
2828                         if [[ "${remote_target}" == *: ]]; then
2829                                 remote_target=${remote_target}${WNAME}
2830                         fi
2831                 else
2832                         if [[ ${target_no_prefix} == */* ]]; then
2833                                 print "ERROR: badly formed upload URI" \
2834                                     "($remote_target)"
2835                                 exit 1
2836                         else
2837                                 remote_target=${remote_target}:${WNAME}
2838                         fi
2839                 fi
2840         fi

2842         #
2843         # Strip trailing slash. Each upload method will deal with directory
2844         # specification separately.
2845         #
2846         remote_target=${remote_target%/}
2847 fi

2849 #
2850 # Option -D by itself (option -U not present) implies no webrev generation.
2851 #
2852 if [[ -z $Uflag && -n $Dflag ]]; then
2853         delete_webrev 1 1
2854         exit $?
2855 fi

2857 #
2858 # Do not generate the webrev, just upload it or delete it.
2859 #
2860 if [[ -n $nflag ]]; then
2861         if [[ -n $Dflag ]]; then
2862                 delete_webrev 1 1
2863                 (( $? == 0 )) || exit $?
2864         fi
2865         if [[ -n $Uflag ]]; then
2866                 upload_webrev
2867                 exit $?
2868         fi
```



```
2869 fi

2871 if [ "${WDIR%%/*}" ]; then
2872         WDIR=$PWD/$WDIR
2873 fi

2875 if [[ ! -d $WDIR ]]; then
2876         mkdir -p $WDIR
2877         (( $? != 0 )) && exit 1
2878 fi

2880 #
2881 # Summarize what we're going to do.
2882 #
2883 print "      Workspace: ${PRETTY_CWS:-$CWS}"
2884 if [[ -n $parent_webrev ]]; then
2885         print "Compare against: webrev at $parent_webrev"
2886 else
2887         print "Compare against: ${PRETTY_PWS:-$PWS}"
2888 fi

2890 [[ -n $INCLUDE_FILE ]] && print "      Including: $INCLUDE_FILE"
2891 print "      Output to: $WDIR"

2893 #
2894 # Save the file list in the webrev dir
2895 #
2896 [[ ! $FLIST -ef $WDIR/file.list ]] && cp $FLIST $WDIR/file.list

2898 rm -f $WDIR/$WNAME.patch
2899 rm -f $WDIR/$WNAME.ps
2900 rm -f $WDIR/$WNAME.pdf

2902 touch $WDIR/$WNAME.patch

2904 print "   Output Files:"

2906 #
2907 # Clean up the file list: Remove comments, blank lines and env variables.
2908 #
2909 $SED -e "s/#.*$//" -e "/=/d" -e "/^[   ]*$/d" $FLIST > /tmp/$$.flist.clean
2910 FLIST=/tmp/$$.flist.clean

2912 #
2913 # For Mercurial, create a cache of manifest entries.
2914 #
2915 if [[ $SCM_MODE == "mercurial" ]]; then
2916         #
2917         # Transform the FLIST into a temporary sed script that matches
2918         # relevant entries in the Mercurial manifest as follows:
2919         # 1) The script will be used against the parent revision manifest,
2920         #    so for FLIST lines that have two filenames (a renamed file)
2921         #    keep only the old name.
2922         # 2) Escape all forward slashes the filename.
2923         # 3) Change the filename into another sed command that matches
2924         #    that file in "hg manifest -v" output:  start of line, three
2925         #    octal digits for file permissions, space, a file type flag
2926         #    character, space, the filename, end of line.
2927         # 4) Eliminate any duplicate entries.  (This can occur if a
2928         #    file has been used as the source of an hg cp and it's
2929         #    also been modified in the same changeset.)
2930         #
2931         SEDFILE=/tmp/$$.manifest.sed
2932         $SED '
2933                 s#^[^ ]* ##
2934                 s#/#\\\/#g
```

```
2935                 s#^.*$#/^... . &$/p#
2936         ' < $FLIST | $SORT -u > $SEDFILE

2938         #
2939         # Apply the generated script to the output of "hg manifest -v"
2940         # to get the relevant subset for this webrev.
2941         #
2942         HG_PARENT_MANIFEST=/tmp/$$.manifest
2943         hg -R $CWS manifest -v -r $HG_PARENT |
2944             $SED -n -f $SEDFILE > $HG_PARENT_MANIFEST
2945 fi

2947 #
2948 # First pass through the files: generate the per-file webrev HTML-files.
2949 #
2950 cat $FLIST | while read LINE
2951 do
2952         set - $LINE
2953         P=$1

2955         #
2956         # Normally, each line in the file list is just a pathname of a
2957         # file that has been modified or created in the child.  A file
2958         # that is renamed in the child workspace has two names on the
2959         # line: new name followed by the old name.
2960         #
2961         oldname=""
2962         oldpath=""
2963         rename=
2964         if [[ $# -eq 2 ]]; then
2965                 PP=$2                   # old filename
2966                 if [[ -f $PP ]]; then
2967                         oldname=" (copied from $PP)"
2968                 else
2969                         oldname=" (renamed from $PP)"
2970                 fi
2971                 oldpath="$PP"
2972                 rename=1
2973                 PDIR=${PP%/*}
2974                 if [[ $PDIR == $PP ]]; then
2975                         PDIR="."   # File at root of workspace
2976                 fi

2978                 PF=${PP##*/}

2980                 DIR=${P%/*}
2981                 if [[ $DIR == $P ]]; then
2982                         DIR="."   # File at root of workspace
2983                 fi

2985                 F=${P##*/}

2987         else
2988                 DIR=${P%/*}
2989                 if [[ "$DIR" == "$P" ]]; then
2990                         DIR="."   # File at root of workspace
2991                 fi

2993                 F=${P##*/}

2995                 PP=$P
2996                 PDIR=$DIR
2997                 PF=$F
2998         fi

3000         COMM=‘getcomments html $P $PP‘
```



```
3002         print "\t$P$oldname\n\t\t\c"

3004         # Make the webrev mirror directory if necessary
3005         mkdir -p $WDIR/$DIR

3007         #
3008         # We stash old and new files into parallel directories in $WDIR
3009         # and do our diffs there.  This makes it possible to generate
3010         # clean looking diffs which don’t have absolute paths present.
3011         #

3013         build_old_new "$WDIR" "$PWS" "$PDIR" "$PF" "$CWS" "$DIR" "$F" || \
3014             continue

3016         #
3017         # Keep the old PWD around, so we can safely switch back after
3018         # diff generation, such that build_old_new runs in a
3019         # consistent environment.
3020         #
3021         OWD=$PWD
3022         cd $WDIR/raw_files
3023         ofile=old/$PDIR/$PF
3024         nfile=new/$DIR/$F

3026         mv_but_nodiff=
3027         cmp $ofile $nfile > /dev/null 2>&1
3028         if [[ $? == 0 && $rename == 1 ]]; then
3029                 mv_but_nodiff=1
3030         fi

3032         #
3033         # If we have old and new versions of the file then run the appropriate
3034         # diffs.  This is complicated by a couple of factors:
3035         #
3036         #       - renames must be handled specially: we emit a ’remove’
3037         #         diff and an ’add’ diff
3038         #       - new files and deleted files must be handled specially
3039         #       - Solaris patch(1m) can’t cope with file creation
3040         #         (and hence renames) as of this writing.
3041         #       - To make matters worse, gnu patch doesn’t interpret the
3042         #         output of Solaris diff properly when it comes to
3043         #         adds and deletes.  We need to do some "cleansing"
3044         #         transformations:
3045         #           [to add a file] @@ -1,0 +X,Y @@  -->  @@ -0,0 +X,Y @@
3046         #           [to del a file] @@ -X,Y +1,0 @@  -->  @@ -X,Y +0,0 @@
3047         #
3048         cleanse_rmfile="$SED ’s/^\(@@ [0-9+,-]*\) [0-9+,-]* @@$/\1 +0,0 @@/’"
3049         cleanse_newfile="$SED ’s/^@@ [0-9+,-]* \([0-9+,-]* @@\)$/@@ -0,0 \1/’"

3051         rm -f $WDIR/$DIR/$F.patch
3052         if [[ -z $rename ]]; then
3053                 if [ ! -f "$ofile" ]; then
3054                         diff -u /dev/null $nfile | sh -c "$cleanse_newfile" \
3055                             > $WDIR/$DIR/$F.patch
3056                 elif [ ! -f "$nfile" ]; then
3057                         diff -u $ofile /dev/null | sh -c "$cleanse_rmfile" \
3058                             > $WDIR/$DIR/$F.patch
3059                 else
3060                         diff -u $ofile $nfile > $WDIR/$DIR/$F.patch
3061                 fi
3062         else
3063                 diff -u $ofile /dev/null | sh -c "$cleanse_rmfile" \
3064                     > $WDIR/$DIR/$F.patch

3066                 diff -u /dev/null $nfile | sh -c "$cleanse_newfile" \
```

```
3067                        >> $WDIR/$DIR/$F.patch
3068        fi

3070        #
3071        # Tack the patch we just made onto the accumulated patch for the
3072        # whole wad.
3073        #
3074        cat $WDIR/$DIR/$F.patch >> $WDIR/$WNAME.patch

3076        print " patch\c"

3078        if [[ -f $ofile && -f $nfile && -z $mv_but_nodiff ]]; then

3080                ${CDIFFCMD:-diff -bt -C 5} $ofile $nfile > $WDIR/$DIR/$F.cdiff
3081                diff_to_html $F $DIR/$F "C" "$COMM" < $WDIR/$DIR/$F.cdiff \
3082                    > $WDIR/$DIR/$F.cdiff.html
3083                print " cdiffs\c"

3085                ${UDIFFCMD:-diff -bt -U 5} $ofile $nfile > $WDIR/$DIR/$F.udiff
3086                diff_to_html $F $DIR/$F "U" "$COMM" < $WDIR/$DIR/$F.udiff \
3087                    > $WDIR/$DIR/$F.udiff.html

3089                print " udiffs\c"

3091                if [[ -x $WDIFF ]]; then
3092                        $WDIFF -c "$COMM" \
3093                            -t "$WNAME Wdiff $DIR/$F" $ofile $nfile > \
3094                            $WDIR/$DIR/$F.wdiff.html 2>/dev/null
3095                        if [[ $? -eq 0 ]]; then
3096                                print " wdiffs\c"
3097                        else
3098                                print " wdiffs[fail]\c"
3099                        fi
3100                fi

3102                sdiff_to_html $ofile $nfile $F $DIR "$COMM" \
3103                    > $WDIR/$DIR/$F.sdiff.html
3104                print " sdiffs\c"

3106                print " frames\c"

3108                rm -f $WDIR/$DIR/$F.cdiff $WDIR/$DIR/$F.udiff

3110                difflines $ofile $nfile > $WDIR/$DIR/$F.count

3112        elif [[ -f $ofile && -f $nfile && -n $mv_but_nodiff ]]; then
3113                # renamed file: may also have differences
3114                difflines $ofile $nfile > $WDIR/$DIR/$F.count
3115        elif [[ -f $nfile ]]; then
3116                # new file: count added lines
3117                difflines /dev/null $nfile > $WDIR/$DIR/$F.count
3118        elif [[ -f $ofile ]]; then
3119                # old file: count deleted lines
3120                difflines $ofile /dev/null > $WDIR/$DIR/$F.count
3121        fi

3123        #
3124        # Now we generate the postscript for this file.  We generate diffs
3125        # only in the event that there is delta, or the file is new (it seems
3126        # tree-killing to print out the contents of deleted files).
3127        #
3128        if [[ -f $nfile ]]; then
3129                ocr=$ofile
3130                [[ ! -f $ofile ]] && ocr=/dev/null

3132                if [[ -z $mv_but_nodiff ]]; then
```



```
3133                        textcomm=‘getcomments text $P $PP‘
3134                        if [[ -x $CODEREVIEW ]]; then
3135                                $CODEREVIEW -y "$textcomm" \
3136                                    -e $ocr $nfile \
3137                                    > /tmp/$$.psfile 2>/dev/null &&
3138                                    cat /tmp/$$.psfile >> $WDIR/$WNAME.ps
3139                                if [[ $? -eq 0 ]]; then
3140                                        print " ps\c"
3141                                else
3142                                        print " ps[fail]\c"
3143                                fi
3144                        fi
3145                fi
3146        fi

3148        if [[ -f $ofile ]]; then
3149                source_to_html Old $PP < $ofile > $WDIR/$DIR/$F-.html
3150                print " old\c"
3151        fi

3153        if [[ -f $nfile ]]; then
3154                source_to_html New $P < $nfile > $WDIR/$DIR/$F.html
3155                print " new\c"
3156        fi

3158        cd $OWD

3160        print
3161 done

3163 frame_nav_js > $WDIR/ancnav.js
3164 frame_navigation > $WDIR/ancnav.html

3166 if [[ ! -f $WDIR/$WNAME.ps ]]; then
3167         print " Generating PDF: Skipped: no output available"
3168 elif [[ -x $CODEREVIEW && -x $PS2PDF ]]; then
3169         print " Generating PDF: \c"
3170         fix_postscript $WDIR/$WNAME.ps | $PS2PDF - > $WDIR/$WNAME.pdf
3171         print "Done."
3172 else
3173         print " Generating PDF: Skipped: missing ’ps2pdf’ or ’codereview’"
3174 fi

3176 # If we’re in OpenSolaris mode and there’s a closed dir under $WDIR,
3177 # delete it - prevent accidental publishing of closed source

3179 if [[ -n "$Oflag" ]]; then
3180         $FIND $WDIR -type d -name closed -exec /bin/rm -rf {} \;
3181 fi

3183 # Now build the index.html file that contains
3184 # links to the source files and their diffs.

3186 cd $CWS

3188 # Save total changed lines for Code Inspection.
3189 print "$TOTL" > $WDIR/TotalChangedLines

3191 print "     index.html: \c"
3192 INDEXFILE=$WDIR/index.html
3193 exec 3<&1                       # duplicate stdout to FD3.
3194 exec 1<&-                       # Close stdout.
3195 exec > $INDEXFILE               # Open stdout to index file.

3197 print "$HTML<head>$STDHEAD"
3198 print "<title>$WNAME</title>"
```

```
3199 print "</head>"
3200 print "<body id=\"SUNWwebrev\">"
3201 print "<div class=\"summary\">"
3202 print "<h2>Code Review for $WNAME</h2>"

3204 print "<table>"

3206 #
3207 # Get the preparer's name:
3208 #
3209 # If the SCM detected is Mercurial, and the configuration property
3210 # ui.username is available, use that, but be careful to properly escape
3211 # angle brackets (HTML syntax characters) in the email address.
3212 #
3213 # Otherwise, use the current userid in the form "John Doe (jdoe)", but
3214 # to maintain compatibility with passwd(4), we must support '&' substitutions.
3215 #
3216 preparer=
3217 if [[ "$SCM_MODE" == mercurial ]]; then
3218         preparer='hg showconfig ui.username 2>/dev/null'
3219         if [[ -n "$preparer" ]]; then
3220                 preparer="$(echo "$preparer" | html_quote)"
3221         fi
3222 fi
3223 if [[ -z "$preparer" ]]; then
3224         preparer=$(
3225             $PERL -e '
3226                 ($login, $pw, $uid, $gid, $quota, $cmt, $gcos) = getpwuid($<);
3227                 if ($login) {
3228                     $gcos =~ s/\&/ucfirst($login)/e;
3229                     printf "%s (%s)\n", $gcos, $login;
3230                 } else {
3231                     printf "(unknown)\n";
3232                 }
3233         ')
3234 fi

3236 PREPDATE=$(LC_ALL=C /usr/bin/date +%Y-%b-%d\ %R\ %z\ %Z)
3237 print "<tr><th>Prepared by:</th><td>$preparer on $PREPDATE</td></tr>"
3238 print "<tr><th>Workspace:</th><td>${PRETTY_CWS:-$CWS}"
3239 print "</td></tr>"
3240 print "<tr><th>Compare against:</th><td>"
3241 if [[ -n $parent_webrev ]]; then
3242         print "webrev at $parent_webrev"
3243 else
3244         print "${PRETTY_PWS:-$PWS}"
3245 fi
3246 print "</td></tr>"
3247 print "<tr><th>Summary of changes:</th><td>"
3248 printCI $TOTL $TINS $TDEL $TMOD $TUNC
3249 print "</td></tr>"

3251 if [[ -f $WDIR/$WNAME.patch ]]; then
3252         wpatch_url="$(print $WNAME.patch | url_encode)"
3253         print "<tr><th>Patch of changes:</th><td>"
3254         print "<a href=\"$wpatch_url\">$WNAME.patch</a></td></tr>"
3255 fi
3256 if [[ -f $WDIR/$WNAME.pdf ]]; then
3257         wpdf_url="$(print $WNAME.pdf | url_encode)"
3258         print "<tr><th>Printable review:</th><td>"
3259         print "<a href=\"$wpdf_url\">$WNAME.pdf</a></td></tr>"
3260 fi

3262 if [[ -n "$iflag" ]]; then
3263         print "<tr><th>Author comments:</th><td><div>"
3264         cat /tmp/$$.include
```



```
3265         print "</div></td></tr>"
3266 fi
3267 print "</table>"
3268 print "</div>"

3270 #
3271 # Second pass through the files: generate the rest of the index file
3272 #
3273 cat $FLIST | while read LINE
3274 do
3275         set - $LINE
3276         P=$1

3278         if [[ $# == 2 ]]; then
3279                 PP=$2
3280                 oldname="$PP"
3281         else
3282                 PP=$P
3283                 oldname=""
3284         fi

3286         mv_but_nodiff=
3287         cmp $WDIR/raw_files/old/$PP $WDIR/raw_files/new/$P > /dev/null 2>&1
3288         if [[ $? == 0 && -n "$oldname" ]]; then
3289                 mv_but_nodiff=1
3290         fi

3292         DIR=${P%/*}
3293         if [[ $DIR == $P ]]; then
3294                 DIR="."   # File at root of workspace
3295         fi

3297         # Avoid processing the same file twice.
3298         # It's possible for renamed files to
3299         # appear twice in the file list

3301         F=$WDIR/$P

3303         print "<p>"

3305         # If there's a diffs file, make diffs links

3307         if [[ -f $F.cdiff.html ]]; then
3308                 cdiff_url="$(print $P.cdiff.html | url_encode)"
3309                 udiff_url="$(print $P.udiff.html | url_encode)"
3310                 print "<a href=\"$cdiff_url\">Cdiffs</a>"
3311                 print "<a href=\"$udiff_url\">Udiffs</a>"

3313                 if [[ -f $F.wdiff.html && -x $WDIFF ]]; then
3314                         wdiff_url="$(print $P.wdiff.html | url_encode)"
3315                         print "<a href=\"$wdiff_url\">Wdiffs</a>"
3316                 fi

3318                 sdiff_url="$(print $P.sdiff.html | url_encode)"
3319                 print "<a href=\"$sdiff_url\">Sdiffs</a>"

3321                 frames_url="$(print $P.frames.html | url_encode)"
3322                 print "<a href=\"$frames_url\">Frames</a>"
3323         else
3324                 print " ------ ------ ------"

3326                 if [[ -x $WDIFF ]]; then
3327                         print " ------"
3328                 fi

3330                 print " ------"
```

```
3331         fi

3333         # If there's an old file, make the link

3335         if [[ -f $F-.html ]]; then
3336                 oldfile_url="$(print $P-.html | url_encode)"
3337                 print "<a href=\"$oldfile_url\">Old</a>"
3338         else
3339                 print " ---"
3340         fi

3342         # If there's an new file, make the link

3344         if [[ -f $F.html ]]; then
3345                 newfile_url="$(print $P.html | url_encode)"
3346                 print "<a href=\"$newfile_url\">New</a>"
3347         else
3348                 print " ---"
3349         fi

3351         if [[ -f $F.patch ]]; then
3352                 patch_url="$(print $P.patch | url_encode)"
3353                 print "<a href=\"$patch_url\">Patch</a>"
3354         else
3355                 print " -----"
3356         fi

3358         if [[ -f $WDIR/raw_files/new/$P ]]; then
3359                 rawfiles_url="$(print raw_files/new/$P | url_encode)"
3360                 print "<a href=\"$rawfiles_url\">Raw</a>"
3361         else
3362                 print " ---"
3363         fi

3365         print "<b>$P</b>"

3367         # For renamed files, clearly state whether or not they are modified
3368         if [[ -f "$oldname" ]]; then
3369                 if [[ -n "$mv_but_nodiff" ]]; then
3370                         print "<i>(copied from $oldname)</i>"
3371                 else
3372                         print "<i>(copied and modified from $oldname)</i>"
3373                 fi
3374         elif [[ -n "$oldname" ]]; then
3375                 if [[ -n "$mv_but_nodiff" ]]; then
3376                         print "<i>(renamed from $oldname)</i>"
3377                 else
3378                         print "<i>(renamed and modified from $oldname)</i>"
3379                 fi
3380         fi

3382         # If there's an old file, but no new file, the file was deleted
3383         if [[ -f $F-.html && ! -f $F.html ]]; then
3384                 print " <i>(deleted)</i>"
3385         fi

3387         #
3388         # Check for usr/closed and deleted_files/usr/closed
3389         #
3390         if [ ! -z "$Oflag" ]; then
3391                 if [[ $P == usr/closed/* || \
3392                     $P == deleted_files/usr/closed/* ]]; then
3393                         print "  <i>Closed source: omitted from" \
3394                             "this review</i>"
3395                 fi
3396         fi
```



```
3398         print "</p>"
3399         # Insert delta comments

3401         print "<blockquote><pre>"
3402         getcomments html $P $PP
3403         print "</pre>"

3405         # Add additional comments comment

3407         print "<!-- Add comments to explain changes in $P here -->"

3409         # Add count of changes.

3411         if [[ -f $F.count ]]; then
3412             cat $F.count
3413             rm $F.count
3414         fi

3416         if [[ $SCM_MODE == "mercurial" ||
3684         if [[ $SCM_MODE == "teamware" ||
3685             $SCM_MODE == "mercurial" ||
3417             $SCM_MODE == "unknown" ]]; then

3419                 # Include warnings for important file mode situations:
3420                 # 1) New executable files
3421                 # 2) Permission changes of any kind
3422                 # 3) Existing executable files

3424                 old_mode=
3425                 if [[ -f $WDIR/raw_files/old/$PP ]]; then
3426                         old_mode=`get_file_mode $WDIR/raw_files/old/$PP`
3427                 fi

3429                 new_mode=
3430                 if [[ -f $WDIR/raw_files/new/$P ]]; then
3431                         new_mode=`get_file_mode $WDIR/raw_files/new/$P`
3432                 fi

3434                 if [[ -z "$old_mode" && "$new_mode" = *[1357]* ]]; then
3435                         print "<span class=\"chmod\">"
3436                         print "<p>new executable file: mode $new_mode</p>"
3437                         print "</span>"
3438                 elif [[ -n "$old_mode" && -n "$new_mode" &&
3439                     "$old_mode" != "$new_mode" ]]; then
3440                         print "<span class=\"chmod\">"
3441                         print "<p>mode change: $old_mode to $new_mode</p>"
3442                         print "</span>"
3443                 elif [[ "$new_mode" = *[1357]* ]]; then
3444                         print "<span class=\"chmod\">"
3445                         print "<p>executable file: mode $new_mode</p>"
3446                         print "</span>"
3447                 fi
3448         fi

3450         print "</blockquote>"
3451 done

3453 print
3454 print
3455 print "<hr></hr>"
3456 print "<p style=\"font-size: small\">"
3457 print "This code review page was prepared using <b>$0</b>."
3458 print "Webrev is maintained by the <a href=\"http://www.illumos.org\">"
3459 print "illumos</a> project.  The latest version may be obtained"
3460 print "<a href=\"http://src.illumos.org/source/xref/illumos-gate/usr/src/tools/s
```

```
3461 print "</body>"
3462 print "</html>"

3464 exec 1<&-                       # Close FD 1.
3465 exec 1<&3                       # dup FD 3 to restore stdout.
3466 exec 3<&-                       # close FD 3.

3468 print "Done."

3470 #
3471 # If remote deletion was specified and fails do not continue.
3472 #
3473 if [[ -n $Dflag ]]; then
3474         delete_webrev 1 1
3475         (( $? == 0 )) || exit $?
3476 fi

3478 if [[ -n $Uflag ]]; then
3479         upload_webrev
3480         exit $?
3481 fi
```