

```

[

{

  "wrong_code":


    """



      #!/bin/sh
      # test-driver - basic testsuite driver script.

      scriptversion=2018-03-07.03; # UTC

      # Copyright (C) 2011-2021 Free Software Foundation, Inc.
      #
      # This program is free software; you can redistribute it and/or modify
      # it under the terms of the GNU General Public License as published by
      # the Free Software Foundation; either version 2, or (at your option)
      # any later version.
      #
      # This program is distributed in the hope that it will be useful,
      # but WITHOUT ANY WARRANTY; without even the implied warranty of
      # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
      # GNU General Public License for more details.
      #
      # You should have received a copy of the GNU General Public License
      # along with this program. If not, see <https://www.gnu.org/licenses/>.

      # As a special exception to the GNU General Public License, if you
      # distribute this file as part of a program that contains a
      # configuration script generated by Autoconf, you may include it under
      # the same distribution terms that you use for the rest of that program.

      # This file is maintained in Automake, please report
      # bugs to <bug-automake@gnu.org> or send patches to
      # <automake-patches@gnu.org>.

      # Make unconditional expansion of undefined variables an error. This
      # helps a lot in preventing typo-related bugs.
      set -u

      usage_error ()
      {
        echo "$0: $*" >&2
        print_usage >&2
        exit 2
      }

      print_usage ()
      {
        cat <<END
        Usage:
        test-driver --test-name NAME --log-file PATH --trs-file PATH
          [--expect-failure {yes|no}] [--color-tests {yes|no}]
          [--enable-hard-errors {yes|no}] [--]
          TEST-SCRIPT [TEST-SCRIPT-ARGUMENTS]
      
```

The '--test-name', '--log-file' and '--trs-file' options are mandatory.  
 See the GNU Automake documentation for information.

END  
{

test\_name= # Used for reporting.  
log\_file= # Where to save the output of the test script.

```

trs_file= # Where to save the metadata of the test run.
expect_failure=no
color_tests=no
enable_hard_errors=yes
while test $# -gt 0; do
case $1 in
--help) print_usage; exit $?;;
--version) echo "test-driver $scriptversion"; exit $?;;
--test-name) test_name=$2; shift;;
--log-file) log_file=$2; shift;;
--trs-file) trs_file=$2; shift;;
--color-tests) color_tests=$2; shift;;
--expect-failure) expect_failure=$2; shift;;
--enable-hard-errors) enable_hard_errors=$2; shift;;
--) shift; break;;
-*) usage_error "invalid option: '$1'";;
*) break;;
esac
shift
done

missing_opts=
test x"$test_name" = x && missing_opts="$missing_opts --test-name"
test x"$log_file" = x && missing_opts="$missing_opts --log-file"
test x"$trs_file" = x && missing_opts="$missing_opts --trs-file"
if test x"$missing_opts" != x; then
usage_error "the following mandatory options are missing:$missing_opts"
fi

if test $# -eq 0; then
usage_error "missing argument"
fi

if test $color_tests = yes; then
# Keep this in sync with 'lib/am/check.am:$(am__tty_colors)'.
red='' # Red.
grn='' # Green.
lgn='' # Light green.
blu='' # Blue.
mgn='' # Magenta.
std='' # No color.
else
red= grn= lgn= blu= mgn= std=
fi

do_exit='rm -f $log_file $trs_file; (exit $st); exit $st'
trap "st=129; $do_exit" 1
trap "st=130; $do_exit" 2
trap "st=141; $do_exit" 13
trap "st=143; $do_exit" 15

# Test script is run here. We create the file first, then append to it,
# to ameliorate tests themselves also writing to the log file. Our tests
# don't, but others can (automake bug#35762).
: >"$log_file"
"$@" >>"$log_file" 2>&1
estatus=$?

if test $enable_hard_errors = no && test $estatus -eq 99; then
tweaked_estatus=1

```

```

else
tweaked_estatus=$estatus
fi

case $tweaked_estatus:$expect_failure in
0:yes) col=$red res=XPASS recheck=yes gcopy=yes;;
0:*) col=$grn res=PASS recheck=no gcopy=no;;
77:*) col=$blu res=SKIP recheck=no gcopy=yes;;
99:*) col=$mgn res=ERROR recheck=yes gcopy=yes;;
*:yes) col=$lgn res=XFAIL recheck=no gcopy=yes;;
*:*) col=$red res=FAIL recheck=yes gcopy=yes;;
esac

# Report the test outcome and exit status in the logs, so that one can
# know whether the test passed or failed simply by looking at the '.log'
# file, without the need of also peaking into the corresponding '.trs'
# file (automake bug#11814).
echo "$res $test_name (exit status: $estatus)" >"$log_file"

# Report outcome to console.
echo "${col}${res}${std}: $test_name"

# Register the test result, and other relevant metadata.
echo ":test-result: $res" > $trs_file
echo ":global-test-result: $res" >> $trs_file
echo ":recheck: $recheck" >> $trs_file
echo ":copy-in-global-log: $gcopy" >> $trs_file

# Local Variables:
# mode: shell-script
# sh-indentation: 2
# eval: (add-hook 'before-save-hook 'time-stamp)
# time-stamp-start: "scriptversion="
# time-stamp-format: "%:y-%02m-%02d.%02H"
# time-stamp-time-zone: "UTC0"
# time-stamp-end: ";" # UTC
# End:
"""
"error_category":"signal handling issues",
"error":"trap: invalid signal specification '13'",
"correct_code":
"""

#!/bin/sh
# test-driver - basic testsuite driver script.

scriptversion=2018-03-07.03; # UTC

# Copyright (C) 2011-2021 Free Software Foundation, Inc.
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2, or (at your option)
# any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License

```

```

# along with this program. If not, see <https://www.gnu.org/licenses/>.

# As a special exception to the GNU General Public License, if you
# distribute this file as part of a program that contains a
# configuration script generated by Autoconf, you may include it under
# the same distribution terms that you use for the rest of that program.

# This file is maintained in Automake, please report
# bugs to <bug-automake@gnu.org> or send patches to
# <automake-patches@gnu.org>.

# Make unconditional expansion of undefined variables an error. This
# helps a lot in preventing typo-related bugs.
set -u

usage_error ()
{
echo "$0: $*" >&2
print_usage >&2
exit 2
}

print_usage ()
{
cat <<END
Usage:
test-driver --test-name NAME --log-file PATH --trs-file PATH
  [--expect-failure {yes|no}] [--color-tests {yes|no}]
  [--enable-hard-errors {yes|no}] [--]
  TEST-SCRIPT [TEST-SCRIPT-ARGUMENTS]

The '--test-name', '--log-file' and '--trs-file' options are mandatory.
See the GNU Automake documentation for information.
END
}

test_name= # Used for reporting.
log_file= # Where to save the output of the test script.
trs_file= # Where to save the metadata of the test run.
expect_failure=no
color_tests=no
enable_hard_errors=yes
while test $# -gt 0; do
case $1 in
--help) print_usage; exit $?;;
--version) echo "test-driver $scriptversion"; exit $?;;
--test-name) test_name=$2; shift;;
--log-file) log_file=$2; shift;;
--trs-file) trs_file=$2; shift;;
--color-tests) color_tests=$2; shift;;
--expect-failure) expect_failure=$2; shift;;
--enable-hard-errors) enable_hard_errors=$2; shift;;
--) shift; break;;
-* ) usage_error "invalid option: '$1'";;
*) break;;
esac
shift
done

missing_opts=

```

```

test x "$test_name" = x && missing_opts="$missing_opts --test-name"
test x "$log_file" = x && missing_opts="$missing_opts --log-file"
test x "$trs_file" = x && missing_opts="$missing_opts --trs-file"
if test x "$missing_opts" != x; then
  usage_error "the following mandatory options are missing:$missing_opts"
fi

if test $# -eq 0; then
  usage_error "missing argument"
fi

if test $color_tests = yes; then
  # Keep this in sync with 'lib/am/check.am:$(am__tty_colors)'.
  red="" # Red.
  grn="" # Green.
  lgn="" # Light green.
  blu="" # Blue.
  mgn="" # Magenta.
  std="" # No color.
  else
    red= grn= lgn= blu= mgn= std=
  fi

do_exit='rm -f $log_file $trs_file; (exit $st); exit $st'
trap "st=129; $do_exit" 1
trap "st=130; $do_exit" 2
trap "st=141; $do_exit" SIGPIPE
trap "st=143; $do_exit" 15

# Test script is run here. We create the file first, then append to it,
# to ameliorate tests themselves also writing to the log file. Our tests
# don't, but others can (automake bug#35762).
: >"$log_file"
"$@" >>"$log_file" 2>&1
estatus=$?

if test $enable_hard_errors = no && test $estatus -eq 99; then
  tweaked_estatus=1
else
  tweaked_estatus=$estatus
fi

case $tweaked_estatus:$expect_failure in
  0:yes) col=$red res=XPASS recheck=yes gcopy=yes;;
  0:*) col=$grn res=PASS recheck=no gcopy=no;;
  77:*) col=$blu res=SKIP recheck=no gcopy=yes;;
  99:*) col=$mgn res=ERROR recheck=yes gcopy=yes;;
  *:yes) col=$lgn res=XFAIL recheck=no gcopy=yes;;
  *:*) col=$red res=FAIL recheck=yes gcopy=yes;;
esac

# Report the test outcome and exit status in the logs, so that one can
# know whether the test passed or failed simply by looking at the '.log'
# file, without the need of also peaking into the corresponding '.trs'
# file (automake bug#11814).
echo "$res $test_name (exit status: $estatus)" >>"$log_file"

# Report outcome to console.
echo "${col}${res}${std}: $test_name"

```

```

# Register the test result, and other relevant metadata.
echo ":test-result: $res" > $trs_file
echo ":global-test-result: $res" >> $trs_file
echo ":recheck: $recheck" >> $trs_file
echo ":copy-in-global-log: $gcopy" >> $trs_file

# Local Variables:
# mode: shell-script
# sh-indentation: 2
# eval: (add-hook 'before-save-hook 'time-stamp)
# time-stamp-start: "scriptversion="
# time-stamp-format: "%:y-%02m-%02d.%02H"
# time-stamp-time-zone: "UTC0"
# time-stamp-end: "; # UTC"
# End:
"""
"patch":
"""

diff --git a/build-aux/test-driver b/build-aux/test-driver
index be73b80..0321f2a 100755
--- a/build-aux/test-driver
+++ b/build-aux/test-driver
@@ -102,7 +102,7 @@ fi
do_exit='rm -f $log_file $trs_file; (exit $st); exit $st'
trap "st=129; $do_exit" 1
trap "st=130; $do_exit" 2
-trap "st=141; $do_exit" 13
+trap "st=141; $do_exit" SIGPIPE
trap "st=143; $do_exit" 15

# Test script is run here. We create the file first, then append to it,
"""
},

{
  "wrong_code":
"""

/* Traverse a file hierarchy.

Copyright (C) 2004-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */

/*
* Copyright (c) 1990, 1993, 1994
*   The Regents of the University of California. All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions

```

- \* are met:
- \* 1. Redistributions of source code must retain the above copyright
- \* notice, this list of conditions and the following disclaimer.
- \* 2. Redistributions in binary form must reproduce the above copyright
- \* notice, this list of conditions and the following disclaimer in the
- \* documentation and/or other materials provided with the distribution.
- \* 4. Neither the name of the University nor the names of its contributors
- \* may be used to endorse or promote products derived from this software
- \* without specific prior written permission.
- \*
- \* THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND
- \* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
- \* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

PURPOSE

- \* ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE

LIABLE

- \* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
- CONSEQUENTIAL
- \* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE

GOODS

- \* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
- \* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,

STRICT

- \* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY

WAY

- \* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
- \* SUCH DAMAGE.
- \*/

```
#include <config.h>

#if defined LIBC_SCCS && !defined GCC_LINT && !defined lint
static char sccsid[] = "@(#)fts.c      8.6 (Berkeley) 8/14/94";
#endif

#include "fts_.h"

#if HAVE_SYS_PARAM_H || defined _LIBC
# include <sys/param.h>
#endif
#ifndef _LIBC
# include <include/sys/stat.h>
#else
# include <sys/stat.h>
#endif
#include <fcntl.h>
#include <errno.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#if !_LIBC
# include "attribute.h"
# include "fcntl--.h"
# include "flexmember.h"
# include "openat.h"
# include "opendirat.h"
# include "same-inode.h"
```

```

#endif

#include <dirent.h>
#ifndef _D_EXACT_NAMLEN
#define _D_EXACT_NAMLEN(dirent) strlen ((dirent)->d_name)
#endif

#if HAVE_STRUCT_DIRENT_D_TYPE
/* True if the type of the directory entry D is known. */
#define DT_IS_KNOWN(d) ((d)->d_type != DT_UNKNOWN)
/* True if the type of the directory entry D must be T. */
#define DT_MUST_BE(d, t) ((d)->d_type == (t))
#define D_TYPE(d) ((d)->d_type)
#else
#define DT_IS_KNOWN(d) false
#define DT_MUST_BE(d, t) false
#define D_TYPE(d) DT_UNKNOWN

#undef DT_UNKNOWN
#define DT_UNKNOWN 0

/* Any nonzero values will do here, so long as they're distinct.
Undef any existing macros out of the way. */
#undef DT_BLK
#undef DT_CHR
#undef DT_DIR
#undef DT_FIFO
#undef DT_LNK
#undef DT_REG
#undef DT_SOCK
#define DT_BLK 1
#define DT_CHR 2
#define DT_DIR 3
#define DT_FIFO 4
#define DT_LNK 5
#define DT_REG 6
#define DT_SOCK 7
#endif

#ifndef S_IFBLK
#define S_IFBLK 0
#endif
#ifndef S_IFLNK
#define S_IFLNK 0
#endif
#ifndef S_IFSOCK
#define S_IFSOCK 0
#endif

enum
{
NOT_AN_INODE_NUMBER = 0
};

#ifdef D_INO_IN_DIRENT
#define D_INO(dp) (dp)->d_ino
#else
/* Some systems don't have inodes, so fake them to avoid lots of ifdefs. */
#define D_INO(dp) NOT_AN_INODE_NUMBER
#endif

```

```

/* If possible (see max_entries, below), read no more than this many directory
entries at a time. Without this limit (i.e., when using non-NULL
fts_compar), processing a directory with 4,000,000 entries requires ~1GiB
of memory, and handling 64M entries would require 16GiB of memory. */
#ifndef FTS_MAX_READDIR_ENTRIES
#define FTS_MAX_READDIR_ENTRIES 100000
#endif

/* If there are more than this many entries in a directory,
and the conditions mentioned below are satisfied, then sort
the entries on inode number before any further processing. */
#ifndef FTS_INODE_SORT_DIR_ENTRIES_THRESHOLD
#define FTS_INODE_SORT_DIR_ENTRIES_THRESHOLD 10000
#endif

enum
{
    _FTS_INODE_SORT_DIR_ENTRIES_THRESHOLD =
FTS_INODE_SORT_DIR_ENTRIES_THRESHOLD
};

enum Fts_stat
{
    FTS_NO_STAT_REQUIRED = 1,
    FTS_STAT_REQUIRED = 2
};

#ifndef _LIBC
#undef close
#define close __close
#undef closedir
#define closedir __closedir
#undef fchdir
#define fchdir __fchdir
#undef open
#define open __open
#undef readdir
#define readdir __readdir
#else
#undef internal_function
#define internal_function /* empty */
#endif

#ifndef __set_errno
#define __set_errno(Val) errno = (Val)
#endif

/* If this host provides the openat function, then we can avoid
attempting to open "." in some initialization code below. */
#ifndef HAVE_OPENAT
#define HAVE_OPENAT_SUPPORT 1
#else
#define HAVE_OPENAT_SUPPORT 0
#endif

#ifndef NDEBUG
#define fts_assert(expr) ((void) (0 && (expr)))
#else
#define fts_assert(expr) \

```

```

do          \
{
    if (!(expr))      \
        abort();      \
}
while (false)
#endif

#ifndef _LIBC
# if __glibc_has_attribute (__fallthrough__)
# define FALLTHROUGH __attribute__ ((__fallthrough__))
# else
# define FALLTHROUGH ((void) 0)
# endif
#endif

static FTSENT *fts_alloc (FTS *, const char *, size_t) internal_function;
static FTSENT *fts_build (FTS *, int) internal_function;
static void fts_lfree (FTSENT *) internal_function;
static void fts_load (FTS *, FTSENT *) internal_function;
static size_t fts_maxarglen (char * const *) internal_function;
static void fts_padjust (FTS *, FTSENT *) internal_function;
static bool fts_palloc (FTS *, size_t) internal_function;
static FTSENT *fts_sort (FTS *, FTSENT *, size_t) internal_function;
static unsigned short int fts_stat (FTS *, FTSENT *, bool) internal_function;
static int fts_safe_changedir (FTS *, FTSENT *, int, const char *)
    internal_function;

#include "fts-cycle.c"

#ifndef MAX
# define MAX(a,b) ((a) > (b) ? (a) : (b))
#endif

#ifndef SIZE_MAX
# define SIZE_MAX ((size_t) -1)
#endif

#define ISDOT(a)   (a[0] == '.' && (!a[1] || (a[1] == '.' && !a[2])))
#define STREQ(a, b) (strcmp (a, b) == 0)

#define CLR(opt)   (sp->fts_options &= ~(opt))
#define ISSET(opt)  ((sp->fts_options & (opt)) != 0)
#define SET(opt)   (sp->fts_options |= (opt))

/* FIXME: FTS_NOCHDIR is now misnamed.
Call it FTS_USE_FULL_RELATIVE_FILE_NAMES instead. */
#define FCHDIR(sp, fd)           \
(!ISSET(FTS_NOCHDIR) && (ISSET(FTS_CWDFD)           \
? (cwd_advance_fd ((sp), (fd), true), 0) \
: fchdir (fd)))

/* fts_build flags */
/* FIXME: make this an enum */
#define BCHILD     1      /* fts_children */
#define BNAMES     2      /* fts_children, names only */
#define BREAD      3      /* fts_read */

#if GNULIB_FTS_DEBUG

```

```

# include <inttypes.h>
# include <stdio.h>
bool fts_debug = false;
#define Dprintf(x) do { if (fts_debug) printf x; } while (false)
static void fd_ring_check (FTS const *);
static void fd_ring_print (FTS const *, FILE *, char const *);
#else
#define Dprintf(x)
#define fd_ring_check(x)
#define fd_ring_print(a, b, c)
#endif

#define LEAVE_DIR(Fts, Ent, Tag) \
do \
{ \
    Dprintf ((\" %s-leaving: %s\\n\", Tag, (Ent)->fts_path)); \
    leave_dir (Fts, Ent); \
    fd_ring_check (Fts); \
} \
while (false)

static void
fd_ring_clear (I_ring *fd_ring)
{
while ( ! i_ring_empty (fd_ring))
{
    int fd = i_ring_pop (fd_ring);
    if (0 <= fd)
        close (fd);
}
}

/* Overload the fts_statp->st_size member (otherwise unused, when
   fts_info is FTS_NSOK) to indicate whether fts_read should stat
   this entry or not. */
static void
fts_set_stat_required (FTSENT *p, bool required)
{
fts_assert (p->fts_info == FTS_NSOK);
p->fts_statp->st_size = (required
    ? FTS_STAT_REQUIRED
    : FTS_NO_STAT_REQUIRED);
}

/* Virtual fchdir. Advance SP's working directory file descriptor,
   SP->fts_cwd_fd, to FD, and push the previous value onto the fd_ring.
   CHDIR_DOWN_ONE is true if FD corresponds to an entry in the directory
   open on sp->fts_cwd_fd; i.e., to move the working directory one level
   down. */
static void
internal_function
cwd_advance_fd (FTS *sp, int fd, bool chdir_down_one)
{
int old = sp->fts_cwd_fd;
fts_assert (old != fd || old == AT_FDCWD);

if (chdir_down_one)
{
/* Push "old" onto the ring.
   If the displaced file descriptor is non-negative, close it. */

```

```

int prev_fd_in_slot = i_ring_push (&sp->fts_fd_ring, old);
fd_ring_print (sp, stderr, "post-push");
if (0 <= prev_fd_in_slot)
    close (prev_fd_in_slot); /* ignore any close failure */
}
else if ( ! ISSET (FTS_NOCHDIR))
{
if (0 <= old)
    close (old); /* ignore any close failure */
}

sp->fts_cwd_fd = fd;
}

/* Restore the initial, pre-traversal, "working directory".
In FTS_CWDFD mode, we merely call cwd_advance_fd, otherwise,
we may actually change the working directory.
Return 0 upon success. Upon failure, set errno and return nonzero. */
static int
restore_initial_cwd (FTS *sp)
{
int fail = FCHDIR (sp, ISSET (FTS_CWDFD) ? AT_FDCWD : sp->fts_rfd);
fd_ring_clear (&(sp->fts_fd_ring));
return fail;
}

/* Open the directory DIR if possible, and return a file
descriptor. Return -1 and set errno on failure. It doesn't matter
whether the file descriptor has read or write access. */

static int
internal_function
diopen (FTS const *sp, char const *dir)
{
int open_flags = (O_SEARCH | O_CLOEXEC | O_DIRECTORY | O_NOCTTY |
O_NONBLOCK
    | (ISSET (FTS_PHYSICAL) ? O_NOFOLLOW : 0));

int fd = (ISSET (FTS_CWDFD)
    ? openat (sp->fts_cwd_fd, dir, open_flags)
    : open (dir, open_flags));
return fd;
}

FTS *
fts_open (char * const *argv,
register int options,
int (*compar) (FTSENT const **, FTSENT const **))
{
register FTS *sp;
register FTSENT *p, *root;
register size_t nitems;
FTSENT *parent = NULL;
FTSENT *tmp = NULL; /* pacify gcc */
bool defer_stat;

/* Options check. */
if (options & ~FTS_OPTIONMASK) {
    __set_errno (EINVAL);
    return (NULL);
}

```

```

}

if ((options & FTS_NOCHDIR) && (options & FTS_CWDFD)) {
    __set_errno (EINVAL);
    return (NULL);
}
if ( ! (options & (FTS_LOGICAL | FTS_PHYSICAL))) {
    __set_errno (EINVAL);
    return (NULL);
}

/* Allocate/initialize the stream */
sp = calloc (1, sizeof *sp);
if (sp == NULL)
    return (NULL);
sp->fts_compar = compar;
sp->fts_options = options;

/* Logical walks turn on NOCHDIR; symbolic links are too hard. */
if (ISSET(FTS_LOGICAL)) {
    SET(FTS_NOCHDIR);
    CLR(FTS_CWDFD);
}

/* Initialize fts_cwd_fd. */
sp->fts_cwd_fd = AT_FDCWD;
if ( ISSET(FTS_CWDFD) && ! HAVE_OPENAT_SUPPORT)
{
    /* While it isn't technically necessary to open "." this
       early, doing it here saves us the trouble of ensuring
       later (where it'd be messier) that "." can in fact
       be opened. If not, revert to FTS_NOCHDIR mode. */
    int fd = open (".", O_SEARCH | O_CLOEXEC);
    if (fd < 0)
    {
        /* Even if "." is unreadable, don't revert to FTS_NOCHDIR mode
           on systems like Linux+PROC_FS, where our openat emulation
           is good enough. Note: on a system that emulates
           openat via /proc, this technique can still fail, but
           only in extreme conditions, e.g., when the working
           directory cannot be saved (i.e. save_cwd fails) --
           and that happens on Linux only when "." is unreadable
           and the CWD would be longer than PATH_MAX.
           FIXME: once Linux kernel openat support is well established,
           replace the above open call and this entire if/else block
           with the body of the if-block below. */
        if (openat_needs_fchdir ())
        {
            SET(FTS_NOCHDIR);
            CLR(FTS_CWDFD);
        }
    }
    else
    {
        close (fd);
    }
}

/*
 * Start out with 1K of file name space, and enough, in any case,
 * to hold the user's file names.

```

```

        */
#ifndef MAXPATHLEN
#define MAXPATHLEN 1024
#endif
{
    size_t maxarglen = fts_maxarglen(argv);
    if (! fts_palloc(sp, MAX(maxarglen, MAXPATHLEN)))
        goto mem1;
}

/* Allocate/initialize root's parent. */
if (*argv != NULL) {
    if ((parent = fts_alloc(sp, "", 0)) == NULL)
        goto mem2;
    parent->fts_level = FTS_ROOTPARENTLEVEL;
}

/* The classic fts implementation would call fts_stat with
   a new entry for each iteration of the loop below.
   If the comparison function is not specified or if the
   FTS_DEFER_STAT option is in effect, don't stat any entry
   in this loop. This is an attempt to minimize the interval
   between the initial stat/lstat/fstatat and the point at which
   a directory argument is first opened. This matters for any
   directory command line argument that resides on a file system
   without genuine i-nodes. If you specify FTS_DEFER_STAT along
   with a comparison function, that function must not access any
   data via the fts_statp pointer. */
defer_stat = (compar == NULL || ISSET(FTS_DEFER_STAT));

/* Allocate/initialize root(s). */
for (root = NULL, nitems = 0; *argv != NULL; ++argv, ++nitems) {
    /* *Do* allow zero-length file names. */
    size_t len = strlen(*argv);

    if ( !(options & FTS_VERBATIM))
    {
        /* If there are two or more trailing slashes, trim all but one,
           but don't change "//" to "/", and do map "///" to "/". */
        char const *v = *argv;
        if (2 < len && v[len - 1] == '/')
            while (1 < len && v[len - 2] == '/')
                --len;
    }

    if ((p = fts_alloc(sp, *argv, len)) == NULL)
        goto mem3;
    p->fts_level = FTS_ROOTLEVEL;
    p->fts_parent = parent;
    p->fts_accpath = p->fts_name;
    /* Even when defer_stat is true, be sure to stat the first
       command line argument, since fts_read (at least with
       FTS_XDEV) requires that. */
    if (defer_stat && root != NULL) {
        p->fts_info = FTS_NSOK;
        fts_set_stat_required(p, true);
    } else {
        p->fts_info = fts_stat(sp, p, false);
    }
}

```

```

/*
 * If comparison routine supplied, traverse in sorted
 * order; otherwise traverse in the order specified.
 */
if (compar) {
    p->fts_link = root;
    root = p;
} else {
    p->fts_link = NULL;
    if (root == NULL)
        tmp = root = p;
    else {
        tmp->fts_link = p;
        tmp = p;
    }
}
if (compar && nitems > 1)
    root = fts_sort(sp, root, nitems);

/*
 * Allocate a dummy pointer and make fts_read think that we've just
 * finished the node before the root(s); set p->fts_info to FTS_INIT
 * so that everything about the "current" node is ignored.
 */
if ((sp->fts_cur = fts_alloc(sp, "", 0)) == NULL)
    goto mem3;
sp->fts_cur->fts_link = root;
sp->fts_cur->fts_info = FTS_INIT;
sp->fts_cur->fts_level = 1;
if (! setup_dir (sp))
    goto mem3;

/*
 * If using chdir(2), grab a file descriptor pointing to dot to ensure
 * that we can get back here; this could be avoided for some file names,
 * but almost certainly not worth the effort. Slashes, symbolic links,
 * and ".." are all fairly nasty problems. Note, if we can't get the
 * descriptor we run anyway, just more slowly.
 */
if (!ISSET(FTS_NOCHDIR) && !ISSET(FTS_CWDFD)
    && (sp->fts_rfd = diropen (sp, ".")) < 0)
    SET(FTS_NOCHDIR);

i_ring_init (&sp->fts_fd_ring, -1);
return (sp);

mem3: fts_lfree(root);
    free(parent);
mem2: free(sp->fts_path);
mem1: free(sp);
    return (NULL);
}

static void
internal_function
fts_load (FTS *sp, register FTSENT *p)
{
    register size_t len;
    register char *cp;

```

```

/*
 * Load the stream structure for the next traversal. Since we don't
 * actually enter the directory until after the preorder visit, set
 * the fts_accpath field specially so the chdir gets done to the right
 * place and the user can access the first node. From fts_open it's
 * known that the file name will fit.
 */
len = p->fts_pathlen = p->fts_namelen;
memmove(sp->fts_path, p->fts_name, len + 1);
if ((cp = strrchr(p->fts_name, '/')) && (cp != p->fts_name || cp[1])) {
    len = strlen(++cp);
    memmove(p->fts_name, cp, len + 1);
    p->fts_namelen = len;
}
p->fts_accpath = p->fts_path = sp->fts_path;
}

int
fts_close (FTS *sp)
{
    register FTSENT *freep, *p;
    int saved_errno = 0;

    /*
     * This still works if we haven't read anything -- the dummy structure
     * points to the root list, so we step through to the end of the root
     * list which has a valid parent pointer.
     */
    if (sp->fts_cur) {
        for (p = sp->fts_cur; p->fts_level >= FTS_ROOTLEVEL;) {
            freep = p;
            p = p->fts_link != NULL ? p->fts_link : p->fts_parent;
            free(freep);
        }
        free(p);
    }

    /* Free up child linked list, sort array, file name buffer. */
    if (sp->fts_child)
        fts_lfree(sp->fts_child);
    free(sp->fts_array);
    free(sp->fts_path);

    if (!ISSET(FTS_CWDFD))
    {
        if (0 <= sp->fts_cwd_fd)
            if (close (sp->fts_cwd_fd))
                saved_errno = errno;
    }
    else if (!ISSET(FTS_NOCHDIR))
    {
        /* Return to original directory, save errno if necessary. */
        if (fchdir(sp->fts_rfd))
            saved_errno = errno;

        /* If close fails, record errno only if saved_errno is zero,
         * so that we report the probably-more-meaningful fchdir errno. */
        if (close (sp->fts_rfd))
            if (saved_errno == 0)

```

```

        saved_errno = errno;
    }

    fd_ring_clear (&sp->fts_fd_ring);

    if (sp->fts_leaf_optimization_works_ht)
        hash_free (sp->fts_leaf_optimization_works_ht);

    free_dir (sp);

    /* Free up the stream pointer. */
    free(sp);

    /* Set errno and return. */
    if (saved_errno) {
        __set_errno (saved_errno);
        return (-1);
    }

    return (0);
}

/* Minimum link count of a traditional Unix directory. When leaf
optimization is OK and a directory's st_nlink == MIN_DIR_NLINK,
then the directory has no subdirectories. */
enum { MIN_DIR_NLINK = 2 };

/* Whether leaf optimization is OK for a directory. */
enum leaf_optimization
{
    /* st_nlink is not reliable for this directory's subdirectories. */
    NO_LEAF_OPTIMIZATION,
    /* st_nlink == 2 means the directory lacks subdirectories. */
    OK_LEAF_OPTIMIZATION
};

#ifndef __linux__ || defined __ANDROID__
&& HAVE_SYS_VFS_H && HAVE_FSTATFS && HAVE_STRUCT_STATFS_F_TYPE
#endif

#include <sys/vfs.h>

/* Linux-specific constants from coreutils' src/fs.h */
#define S_MAGIC_AFS 0x5346414F
#define S_MAGIC_CIFS 0xFF534D42
#define S_MAGIC_NFS 0x6969
#define S_MAGIC_PROC 0x9FA0
#define S_MAGIC_TMPFS 0x1021994

#ifndef HAVE_FSWORD_T
typedef __fsword_t fsword;
#else
typedef long int fsword;
#endif

/* Map a stat.st_dev number to a file system type number f_ftype. */
struct dev_type
{
    dev_t st_dev;
    fsword f_type;
}
```

```

};

/* Use a tiny initial size. If a traversal encounters more than
a few devices, the cost of growing/rehashing this table will be
rendered negligible by the number of inodes processed. */
enum { DEV_TYPE_HT_INITIAL_SIZE = 13 };

static size_t
dev_type_hash (void const *x, size_t table_size)
{
struct dev_type const *ax = x;
uintmax_t dev = ax->st_dev;
return dev % table_size;
}

static bool
dev_type_compare (void const *x, void const *y)
{
struct dev_type const *ax = x;
struct dev_type const *ay = y;
return ax->st_dev == ay->st_dev;
}

/* Return the file system type of P with file descriptor FD, or 0 if not known.
If FD is negative, P's file descriptor is unavailable.
Try to cache known values. */

static fsword
filesystem_type (FTSENT const *p, int fd)
{
FTS *sp = p->fts_fts;
Hash_table *h = sp->fts_leaf_optimization_works_ht;
struct dev_type *ent;
struct statfs fs_buf;

/* If we're not in CWDFD mode, don't bother with this optimization,
   since the caller is not serious about performance. */
if (!ISSET (FTS_CWDFD))
    return 0;

if (!h)
    h = sp->fts_leaf_optimization_works_ht
    = hash_initialize (DEV_TYPE_HT_INITIAL_SIZE, NULL, dev_type_hash,
                       dev_type_compare, free);
if (h)
{
    struct dev_type tmp;
    tmp.st_dev = p->fts_statp->st_dev;
    ent = hash_lookup (h, &tmp);
    if (ent)
        return ent->f_type;
}

/* Look-up failed. Query directly and cache the result. */
if (fd < 0 || fstatfs (fd, &fs_buf) != 0)
    return 0;

if (h)
{
    struct dev_type *t2 = malloc (sizeof *t2);

```

```

if (t2)
{
    t2->st_dev = p->fts_statp->st_dev;
    t2->f_type = fs_buf.f_type;

    ent = hash_insert (h, t2);
    if (ent)
        fts_assert (ent == t2);
    else
        free (t2);
}
}

return fs_buf.f_type;
}

/* Return true if sorting dirents on inode numbers is known to improve
traversal performance for the directory P with descriptor DIR_FD.
Return false otherwise. When in doubt, return true.
DIR_FD is negative if unavailable. */
static bool
dirent_inode_sort_may_be_useful (FTSENT const *p, int dir_fd)
{
/* Skip the sort only if we can determine efficiently
that skipping it is the right thing to do.
The cost of performing an unnecessary sort is negligible,
while the cost of *not* performing it can be O(N^2) with
a very large constant. */

switch (filesystem_type (p, dir_fd))
{
case S_MAGIC_CIFS:
case S_MAGIC_NFS:
case S_MAGIC_TMPFS:
/* On a file system of any of these types, sorting
is unnecessary, and hence wasteful. */
return false;

default:
return true;
}
}

/* Given an FTS entry P for a directory with descriptor DIR_FD,
return whether it is valid to apply leaf optimization.
The optimization is valid if a directory's st_nlink value equal
to MIN_DIR_NLINK means the directory has no subdirectories.
DIR_FD is negative if unavailable. */
static enum leaf_optimization
leaf_optimization (FTSENT const *p, int dir_fd)
{
switch (filesystem_type (p, dir_fd))
{
case 0:
/* Leaf optimization is unsafe if the file system type is unknown. */
FALLTHROUGH;
case S_MAGIC_AFS:
/* Although AFS mount points are not counted in st_nlink, they
act like directories. See <https://bugs.debian.org/143111>. */
FALLTHROUGH;
}
}

```

```

case S_MAGIC_CIFS:
/* Leaf optimization causes 'find' to abort. See
   <https://lists.gnu.org/r/bug-gnulib/2018-04/msg00015.html>. */
FALLTHROUGH;
case S_MAGIC_NFS:
/* NFS provides usable dirent.d_type but not necessarily for all entries
   of large directories, so as per <https://bugzilla.redhat.com/1252549>
   NFS should return true. However st_nlink values are not accurate on
   all implementations as per <https://bugzilla.redhat.com/1299169>. */
FALLTHROUGH;
case S_MAGIC_PROC:
/* Per <https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=143111> /proc
   may have bogus stat.st_nlink values. */
return NO_LEAF_OPTIMIZATION;

default:
return OK_LEAF_OPTIMIZATION;
}

#else
static bool
dirent_inode_sort_may_be_useful (_GL_UNUSED FTSENT const *p,
                                 _GL_UNUSED int dir_fd)
{
return true;
}
static enum leaf_optimization
leaf_optimization (_GL_UNUSED FTSENT const *p, _GL_UNUSED int dir_fd)
{
return NO_LEAF_OPTIMIZATION;
}
#endif

/*
* Special case of "/" at the end of the file name so that slashes aren't
* appended which would cause file names to be written as "....//foo".
*/
#define NAPPEND(p) \
    (p->fts_path[p->fts_pathlen - 1] == '/' \
     ? p->fts_pathlen - 1 : p->fts_pathlen)

FTSENT *
fts_read (register FTS *sp)
{
register FTSENT *p, *tmp;
register unsigned short int instr;
register char *t;

/* If finished or unrecoverable error, return NULL. */
if (sp->fts_cur == NULL || ISSET(FTS_STOP))
    return (NULL);

/* Set current node pointer. */
p = sp->fts_cur;

/* Save and zero out user instructions. */
instr = p->fts_instr;
p->fts_instr = FTS_NOINSTR;

```

```

/* Any type of file may be re-visited; re-stat and re-turn. */
if (instr == FTS_AGAIN) {
    p->fts_info = fts_stat(sp, p, false);
    return (p);
}
Dprintf ("fts_read: p=%s\n",
         p->fts_info == FTS_INIT ? "" : p->fts_path);

/*
 * Following a symlink -- SLNONE test allows application to see
 * SLNONE and recover. If indirecting through a symlink, have
 * keep a pointer to current location. If unable to get that
 * pointer, follow fails.
*/
if (instr == FTS_FOLLOW &&
    (p->fts_info == FTS_SL || p->fts_info == FTS_SLNONE)) {
    p->fts_info = fts_stat(sp, p, true);
    if (p->fts_info == FTS_D && !ISSET(FTS_NOCHDIR)) {
        if ((p->fts_symfd = diropen (sp, ".") < 0) {
            p->fts_errno = errno;
            p->fts_info = FTS_ERR;
        } else
            p->fts_flags |= FTS_SYMFOLLOW;
    }
    goto check_for_dir;
}

/* Directory in pre-order.*/
if (p->fts_info == FTS_D) {
    /* If skipped or crossed mount point, do post-order visit.*/
    if (instr == FTS_SKIP ||
        (ISSET(FTS_XDEV) && p->fts_statp->st_dev != sp->fts_dev)) {
        if (p->fts_flags & FTS_SYMFOLLOW)
            (void)close(p->fts_symfd);
        if (sp->fts_child) {
            fts_lfree(sp->fts_child);
            sp->fts_child = NULL;
        }
        p->fts_info = FTS_DP;
        LEAVE_DIR (sp, p, "1");
        return (p);
    }
}

/* Rebuild if only read the names and now traversing.*/
if (sp->fts_child != NULL && ISSET(FTS_NAMEONLY)) {
    CLR(FTS_NAMEONLY);
    fts_lfree(sp->fts_child);
    sp->fts_child = NULL;
}

/*
 * Cd to the subdirectory.
 *
 * If have already read and now fail to chdir, whack the list
 * to make the names come out right, and set the parent errno
 * so the application will eventually get an error condition.
 * Set the FTS_DONTCHDIR flag so that when we logically change
 * directories back to the parent we don't do a chdir.
 *
 * If haven't read do so. If the read fails, fts_build sets

```

```

        * FTS_STOP or the fts_info field of the node.
    */
    if (sp->fts_child != NULL) {
        if (fts_safe_changedir(sp, p, -1, p->fts_accpath)) {
            p->fts_errno = errno;
            p->fts_flags |= FTS_DONTCHDIR;
            for (p = sp->fts_child; p != NULL;
                 p = p->fts_link)
                p->fts_accpath =
                    p->fts_parent->fts_accpath;
        }
    } else if ((sp->fts_child = fts_build(sp, BREAD)) == NULL) {
        if (ISSET(FTS_STOP))
            return (NULL);
        /* If fts_build's call to fts_safe_changedir failed
         because it was not able to fchdir into a
         subdirectory, tell the caller. */
        if (p->fts_errno && p->fts_info != FTS_DNR)
            p->fts_info = FTS_ERR;
        LEAVE_DIR (sp, p, "2");
        return (p);
    }
    p = sp->fts_child;
    sp->fts_child = NULL;
    goto name;
}

/* Move to the next node on this level. */
next: tmp = p;

/* If we have so many directory entries that we're reading them
in batches, and we've reached the end of the current batch,
read in a new batch. */
if (p->fts_link == NULL && p->fts_parent->fts_dirp)
{
    p = tmp->fts_parent;
    sp->fts_cur = p;
    sp->fts_path[p->fts_pathlen] = '\0';

    if ((p = fts_build (sp, BREAD)) == NULL)
    {
        if (ISSET(FTS_STOP))
            return NULL;
        goto cd_dot_dot;
    }

    free(tmp);
    goto name;
}

if ((p = p->fts_link) != NULL) {
    sp->fts_cur = p;
    free(tmp);

    /*
     * If reached the top, return to the original directory (or
     * the root of the tree), and load the file names for the next
     * root.
     */
    if (p->fts_level == FTS_ROOTLEVEL) {

```

```

        if (restore_initial_cwd(sp)) {
            SET(FTS_STOP);
            return (NULL);
        }
        free_dir(sp);
        fts_load(sp, p);
        if (! setup_dir(sp)) {
            free_dir(sp);
            return (NULL);
        }
        goto check_for_dir;
    }

/*
 * User may have called fts_set on the node. If skipped,
 * ignore. If followed, get a file descriptor so we can
 * get back if necessary.
 */
if (p->fts_instr == FTS_SKIP)
    goto next;
if (p->fts_instr == FTS_FOLLOW) {
    p->fts_info = fts_stat(sp, p, true);
    if (p->fts_info == FTS_D && !ISSET(FTS_NOCHDIR)) {
        if ((p->fts_symfd = diropen (sp, ".")) < 0) {
            p->fts_errno = errno;
            p->fts_info = FTS_ERR;
        } else
            p->fts_flags |= FTS_SYMFOLLOW;
    }
    p->fts_instr = FTS_NOINSTR;
}

name:      t = sp->fts_path + NAPPEND(p->fts_parent);
*t++ = '/';
memmove(t, p->fts_name, p->fts_namelen + 1);
check_for_dir:
    sp->fts_cur = p;
    if (p->fts_info == FTS_NSOK)
    {
        if (p->fts_statp->st_size == FTS_STAT_REQUIRED)
            p->fts_info = fts_stat(sp, p, false);
        else
            fts_assert (p->fts_statp->st_size == FTS_NO_STAT_REQUIRED);
    }

    if (p->fts_info == FTS_D)
    {
        /* Now that P->fts_statp is guaranteed to be valid,
        if this is a command-line directory, record its
        device number, to be used for FTS_XDEV. */
        if (p->fts_level == FTS_ROOTLEVEL)
            sp->fts_dev = p->fts_statp->st_dev;
        Dprintf (" entering: %s\n", p->fts_path);
        if (! enter_dir (sp, p))
            return NULL;
    }
    return p;
}
cd_dot_dot:

```

```

/* Move up to the parent node. */
p = tmp->fts_parent;
sp->fts_cur = p;
free(tmp);

if (p->fts_level == FTS_ROOTPARENTLEVEL) {
    /*
     * Done; free everything up and set errno to 0 so the user
     * can distinguish between error and EOF.
     */
    free(p);
    __set_errno(0);
    return (sp->fts_cur = NULL);
}

fts_assert (p->fts_info != FTS_NSOK);

/* NUL terminate the file name. */
sp->fts_path[p->fts_pathlen] = '\0';

/*
 * Return to the parent directory. If at a root node, restore
 * the initial working directory. If we came through a symlink,
 * go back through the file descriptor. Otherwise, move up
 * one level, via "..".
 */
if (p->fts_level == FTS_ROOTLEVEL) {
    if (restore_initial_cwd(sp)) {
        p->fts_errno = errno;
        SET(FTS_STOP);
    }
} else if (p->fts_flags & FTS_SYMFOLLOW) {
    if (FCHDIR(sp, p->fts_symfd)) {
        p->fts_errno = errno;
        SET(FTS_STOP);
    }
    (void)close(p->fts_symfd);
} else if (!(p->fts_flags & FTS_DONTCHDIR) &&
           fts_safe_changedir(sp, p->fts_parent, -1, "..")) {
    p->fts_errno = errno;
    SET(FTS_STOP);
}

/* If the directory causes a cycle, preserve the FTS_DC flag and keep
   the corresponding dev/ino pair in the hash table. It is going to be
   removed when leaving the original directory. */
if (p->fts_info != FTS_DC) {
    p->fts_info = p->fts_errno ? FTS_ERR : FTS_DP;
    if (p->fts_errno == 0)
        LEAVE_DIR(sp, p, "3");
}
return ISSET(FTS_STOP) ? NULL : p;
}

/*
 * Fts_set takes the stream as an argument although it's not used in this
 * implementation; it would be necessary if anyone wanted to add global
 * semantics to fts using fts_set. An error return is allowed for similar
 * reasons.
*/

```

```

/* ARGSUSED */
int
fts_set(_GL_UNUSED FTS *sp, FTSENT *p, int instr)
{
    if (instr != 0 && instr != FTS_AGAIN && instr != FTS_FOLLOW &&
        instr != FTS_NOINSTR && instr != FTS_SKIP) {
        __set_errno (EINVAL);
        return (1);
    }
    p->fts_instr = instr;
    return (0);
}

FTSENT *
fts_children (register FTS *sp, int instr)
{
    register FTSENT *p;
    int fd;

    if (instr != 0 && instr != FTS_NAMEONLY) {
        __set_errno (EINVAL);
        return (NULL);
    }

    /* Set current node pointer. */
    p = sp->fts_cur;

    /*
     * Errno set to 0 so user can distinguish empty directory from
     * an error.
     */
    __set_errno (0);

    /* Fatal errors stop here. */
    if (ISSET(FTS_STOP))
        return (NULL);

    /* Return logical hierarchy of user's arguments. */
    if (p->fts_info == FTS_INIT)
        return (p->fts_link);

    /*
     * If not a directory being visited in pre-order, stop here. Could
     * allow FTS_DNR, assuming the user has fixed the problem, but the
     * same effect is available with FTS AGAIN.
     */
    if (p->fts_info != FTS_D /* && p->fts_info != FTS_DNR */)
        return (NULL);

    /* Free up any previous child list. */
    if (sp->fts_child != NULL)
        fts_lfree(sp->fts_child);

    if (instr == FTS_NAMEONLY) {
        SET(FTS_NAMEONLY);
        instr = BNAMES;
    } else
        instr = BCHILD;

    /*

```

```

* If using chdir on a relative file name and called BEFORE fts_read
* does its chdir to the root of a traversal, we can lose -- we need to
* chdir into the subdirectory, and we don't know where the current
* directory is, so we can't get back so that the upcoming chdir by
* fts_read will work.
*/
if (p->fts_level != FTS_ROOTLEVEL || p->fts_accpath[0] == '/' ||
    ISSET(FTS_NOCHDIR))
    return (sp->fts_child = fts_build(sp, instr));

if ((fd = diropen (sp, ".") < 0)
    return (sp->fts_child = NULL);
sp->fts_child = fts_build(sp, instr);
if (ISSET(FTS_CWDFD))
{
    cwd_advance_fd (sp, fd, true);
}
else
{
    if (fchdir(fd))
    {
        int saved_errno = errno;
        close (fd);
        __set_errno (saved_errno);
        return NULL;
    }
    close (fd);
}
return (sp->fts_child);
}

/* A comparison function to sort on increasing inode number.
For some file system types, sorting either way makes a huge
performance difference for a directory with very many entries,
but sorting on increasing values is slightly better than sorting
on decreasing values. The difference is in the 5% range. */
static int
fts_compare_ino (struct _ftsent const **a, struct _ftsent const **b)
{
return _GL_CMP (a[0]->fts_statp->st_ino, b[0]->fts_statp->st_ino);
}

/* Map the dirent.d_type value, DTTYPE, to the corresponding stat.st_mode
S_IF* bit and set ST.st_mode, thus clearing all other bits in that field. */
static void
set_stat_type (struct stat *st, unsigned int dtype)
{
mode_t type;
switch (dtype)
{
case DT_BLK:
type = S_IFBLK;
break;
case DT_CHR:
type = S_IFCHR;
break;
case DT_DIR:
type = S_IFDIR;
break;
case DT_FIFO:

```

```

type = S_IFIFO;
break;
case DT_LNK:
type = S_IFLNK;
break;
case DT_REG:
type = S_IFREG;
break;
case DT_SOCK:
type = S_IFSOCK;
break;
default:
type = 0;
}
st->st_mode = type;
}

#define closedir_and_clear(dirp)          \
do {                                       \
    closedir (dirp);                      \
    dirp = NULL;                          \
} while (0)

#define fts_opendir(file, Pdir_fd)        \
opendirat((! ISSET(FTS_NOCHDIR) && ISSET(FTS_CWDFD) \      \
    ? sp->fts_cwd_fd : AT_FDCWD),           \
    file,                                     \
    (((ISSET(FTS_PHYSICAL)                  \
    && ! (ISSET(FTS_COMFOLLOW)            \
    && cur->fts_level == FTS_ROOTLEVEL)) \ \
    ? O_NOFOLLOW : 0)),                     \
    Pdir_fd)

/*
 * This is the tricky part -- do not casually change *anything* in here. The
 * idea is to build the linked list of entries that are used by fts_children
 * and fts_read. There are lots of special cases.
 *
 * The real slowdown in walking the tree is the stat calls. If FTS_NOSTAT is
 * set and it's a physical walk (so that symbolic links can't be directories),
 * we can do things quickly. First, if it's a 4.4BSD file system, the type
 * of the file is in the directory entry. Otherwise, we assume that the number
 * of subdirectories in a node is equal to the number of links to the parent.
 * The former skips all stat calls. The latter skips stat calls in any leaf
 * directories and for any files after the subdirectories in the directory have
 * been found, cutting the stat calls by about 2/3.
 */
static FTSENT *
internal_function
fts_build (register FTS *sp, int type)
{
    register FTSENT *p, *head;
    register size_t nitems;
    FTSENT *tail;
    int saved_errno;
    bool descend;
    bool doadjust;
    ptrdiff_t level;

```

```

size_t len, maxlen, new_len;
char *cp;
int dir_fd;
FTSENT *cur = sp->fts_cur;
bool continue_readdir = !cur->fts_dirp;
bool sort_by_inode = false;
size_t max_entries;

/* When cur->fts_dirp is non-NULL, that means we should
continue calling readdir on that existing DIR* pointer
rather than opening a new one. */
if (continue_readdir)
{
    DIR *dp = cur->fts_dirp;
    dir_fd = dirfd(dp);
    if (dir_fd < 0)
    {
        int dirfd_errno = errno;
        closedir_and_clear(cur->fts_dirp);
        if (type == BREAD)
        {
            cur->fts_info = FTS_DNR;
            cur->fts_errno = dirfd_errno;
        }
        return NULL;
    }
}
else
{
    /* Open the directory for reading. If this fails, we're done.
    If being called from fts_read, set the fts_info field.*/
    if ((cur->fts_dirp = fts_opendir(cur->fts_accpath, &dir_fd)) == NULL)
    {
        if (type == BREAD)
        {
            cur->fts_info = FTS_DNR;
            cur->fts_errno = errno;
        }
        return NULL;
    }
    /* Rather than calling fts_stat for each and every entry encountered
    in the readdir loop (below), stat each directory only right after
    opening it. */
    bool stat_optimization = cur->fts_info == FTS_NSOK;

    if (stat_optimization
        /* Also read the stat info again after opening a directory to
        reveal eventual changes caused by a submount triggered by
        the traversal. But do it only for utilities which use
        FTS_TIGHT_CYCLE_CHECK. Therefore, only find and du
        benefit/suffer from this feature for now. */
        || ISSET(FTS_TIGHT_CYCLE_CHECK))
    {
        if (!stat_optimization)
            LEAVE_DIR(sp, cur, "4");
        if (fstat(dir_fd, cur->fts_statp) != 0)
        {
            int fstat_errno = errno;
            closedir_and_clear(cur->fts_dirp);
            if (type == BREAD)

```

```

    {
        cur->fts_errno = fstat_errno;
        cur->fts_info = FTS_NS;
    }
    __set_errno (fstat_errno);
    return NULL;
}
if (stat_optimization)
cur->fts_info = FTS_D;
else if (! enter_dir (sp, cur))
{
    int err = errno;
    closedir_and_clear (cur->fts_dirp);
    __set_errno (err);
    return NULL;
}
}
}

/* Maximum number of readdir entries to read at one time. This
limitation is to avoid reading millions of entries into memory
at once. When an fts_compar function is specified, we have no
choice: we must read all entries into memory before calling that
function. But when no such function is specified, we can read
entries in batches that are large enough to help us with inode-
sorting, yet not so large that we risk exhausting memory. */
max_entries = sp->fts_compar ? SIZE_MAX : FTS_MAX_READDIR_ENTRIES;

/*
* If we're going to need to stat anything or we want to descend
* and stay in the directory, chdir. If this fails we keep going,
* but set a flag so we don't chdir after the post-order visit.
* We won't be able to stat anything, but we can still return the
* names themselves. Note, that since fts_read won't be able to
* chdir into the directory, it will have to return different file
* names than before, i.e. "a/b" instead of "b". Since the node
* has already been visited in pre-order, have to wait until the
* post-order visit to return the error. There is a special case
* here, if there was nothing to stat then it's not an error to
* not be able to stat. This is all fairly nasty. If a program
* needed sorted entries or stat information, they had better be
* checking FTS_NS on the returned nodes.
*/
if (continue_readdir)
{
    /* When resuming a short readdir run, we already have
     * the required dirp and dir_fd. */
    descend = true;
}
else
{
    /* Try to descend unless it is a names-only fts_children,
     * or the directory is known to lack subdirectories. */
    descend = (type != BNAMES
        && ! (ISSET (FTS_NOSTAT) && ISSET (FTS_PHYSICAL)
        && ! ISSET (FTS_SEEDOT)
        && cur->fts_statp->st_nlink == MIN_DIR_NLINK
        && (leaf_optimization (cur, dir_fd)
            != NO_LEAF_OPTIMIZATION)));
    if (descend || type == BREAD)

```

```

    {
        if (ISSET(FTS_CWDFD))
            dir_fd = fcntl (dir_fd, F_DUPFD_CLOEXEC, STDERR_FILENO + 1);
        if (dir_fd < 0 || fts_safe_changedir(sp, cur, dir_fd, NULL)) {
            if (descend && type == BREAD)
                cur->fts_errno = errno;
            cur->fts_flags |= FTS_DONTCHDIR;
            descend = false;
            closedir_and_clear(cur->fts_dirp);
            if (ISSET(FTS_CWDFD) && 0 <= dir_fd)
                close (dir_fd);
            cur->fts_dirp = NULL;
        } else
            descend = true;
    }
}

/*
 * Figure out the max file name length that can be stored in the
 * current buffer -- the inner loop allocates more space as necessary.
 * We really wouldn't have to do the maxlen calculations here, we
 * could do them in fts_read before returning the name, but it's a
 * lot easier here since the length is part of the dirent structure.
 *
 * If not changing directories set a pointer so that can just append
 * each new component into the file name.
 */
len = NAPPEND(cur);
if (ISSET(FTS_NOCHDIR)) {
    cp = sp->fts_path + len;
    *cp++ = '/';
} else {
    /* GCC, you're too verbose. */
    cp = NULL;
}
len++;
maxlen = sp->fts_pathlen - len;

level = cur->fts_level + 1;

/* Read the directory, attaching each entry to the "link" pointer. */
doadjust = false;
head = NULL;
tail = NULL;
nitems = 0;
while (cur->fts_dirp) {
    size_t d_namelen;
    __set_errno (0);
    struct dirent *dp = readdir(cur->fts_dirp);
    if (dp == NULL) {
        if (errno) {
            cur->fts_errno = errno;
            /* If we've not read any items yet, treat
               the error as if we can't access the dir. */
            cur->fts_info = (continue_readdir || nitems)
                ? FTS_ERR : FTS_DNR;
        }
        closedir_and_clear(cur->fts_dirp);
        break;
    }
}

```

```

if (!ISSET(FTS_SEEDOT) && ISDOT(dp->d_name))
    continue;

d_namelen = _D_EXACT_NAMLEN (dp);
p = fts_alloc (sp, dp->d_name, d_namelen);
if (!p)
    goto mem1;
if (d_namelen >= maxlen) {
    /* include space for NUL */
    uintptr_t oldaddr = (uintptr_t) sp->fts_path;
    if (! fts_palloc(sp, d_namelen + len + 1)) {
        /*
         * No more memory. Save
         * errno, free up the current structure and the
         * structures already allocated.
        */
mem1:           saved_errno = errno;
    free(p);
    fts_lfree(head);
    closedir_and_clear(cur->fts_dirp);
    cur->fts_info = FTS_ERR;
    SET(FTS_STOP);
    __set_errno (saved_errno);
    return (NULL);
    }
    /* Did realloc() change the pointer? */
    if (oldaddr != (uintptr_t) sp->fts_path) {
        doadjust = true;
        if (ISSET(FTS_NOCHDIR))
            cp = sp->fts_path + len;
    }
    maxlen = sp->fts_pathlen - len;
}

new_len = len + d_namelen;
if (new_len < len) {
    /*
     * In the unlikely event that we would end up
     * with a file name longer than SIZE_MAX, free up
     * the current structure and the structures already
     * allocated, then error out with ENAMETOOLONG.
    */
    free(p);
    fts_lfree(head);
    closedir_and_clear(cur->fts_dirp);
    cur->fts_info = FTS_ERR;
    SET(FTS_STOP);
    __set_errno (ENAMETOOLONG);
    return (NULL);
}
p->fts_level = level;
p->fts_parent = sp->fts_cur;
p->fts_pathlen = new_len;

/* Store dirent.d_ino, in case we need to sort
entries before processing them. */
p->fts_statp->st_ino = D_INO (dp);

/* Build a file name for fts_stat to stat. */
if (ISSET(FTS_NOCHDIR)) {

```

```

    p->fts_accpath = p->fts_path;
    memmove(cp, p->fts_name, p->fts_namelen + 1);
} else
    p->fts_accpath = p->fts_name;

if (sp->fts_compar == NULL || ISSET(FTS_DEFER_STAT)) {
    /* Record what fts_read will have to do with this
       entry. In many cases, it will simply fts_stat it,
       but we can take advantage of any d_type information
       to optimize away the unnecessary stat calls. I.e.,
       if FTS_NOSTAT is in effect and we're not following
       symlinks (FTS_PHYSICAL) and d_type indicates this
       is *not* a directory, then we won't have to stat it
       at all. If it *is* a directory, then (currently)
       we stat it regardless, in order to get device and
       inode numbers. Some day we might optimize that
       away, too, for directories where d_ino is known to
       be valid. */
    bool skip_stat = (ISSET(FTS_NOSTAT)
                      && DT_IS_KNOWN(dp)
                      && ! DT_MUST_BE(dp, DT_DIR)
                      && (ISSET(FTS_PHYSICAL)
                           || ! DT_MUST_BE(dp, DT_LNK)));
    p->fts_info = FTS_NSOK;
    /* Propagate dirent.d_type information back
       to caller, when possible. */
    set_stat_type(p->fts_statp, D_TYPE(dp));
    fts_set_stat_required(p, !skip_stat);
} else {
    p->fts_info = fts_stat(sp, p, false);
}

/* We walk in directory order so "ls -f" doesn't get upset. */
p->fts_link = NULL;
if (head == NULL)
    head = tail = p;
else {
    tail->fts_link = p;
    tail = p;
}

/* If there are many entries, no sorting function has been
   specified, and this file system is of a type that may be
   slow with a large number of entries, arrange to sort the
   directory entries on increasing inode numbers.

The NITEMS comparison uses ==, not >, because the test
needs to be tried at most once once, and NITEMS will exceed
the threshold after it is incremented below. */
if (nitems == _FTS_INODE_SORT_DIR_ENTRIES_THRESHOLD
    && !sp->fts_compar)
    sort_by_inode = dirent_inode_sort_may_be_useful(cur, dir_fd);

    ++nitems;
    if (max_entries <= nitems) {
        /* When there are too many dir entries, leave
           fts_dirp open, so that a subsequent fts_read
           can take up where we leave off. */
        break;
}

```

```

    }

/*
 * If realloc() changed the address of the file name, adjust the
 * addresses for the rest of the tree and the dir list.
 */
if (doadjust)
    fts_padjust(sp, head);

/*
 * If not changing directories, reset the file name back to original
 * state.
 */
if (ISSET(FTS_NOCHDIR)) {
    if (len == sp->fts_pathlen || nitems == 0)
        --cp;
    *cp = '\0';
}

/*
 * If descended after called from fts_children or after called from
 * fts_read and nothing found, get back. At the root level we use
 * the saved fd; if one of fts_open()'s arguments is a relative name
 * to an empty directory, we wind up here with no other way back. If
 * can't get back, we're done.
 */
if (!continue_readdir && descend && (type == BCHILD || !nitems) &&
    (cur->fts_level == FTS_ROOTLEVEL
     ? restore_initial_cwd(sp)
     : fts_safe_changecwd(sp, cur->fts_parent, -1, "..")) {
    cur->fts_info = FTS_ERR;
    SET(FTS_STOP);
    fts_lfree(head);
    return (NULL);
}

/* If didn't find anything, return NULL. */
if (!nitems) {
    if (type == BREAD
        && cur->fts_info != FTS_DNR && cur->fts_info != FTS_ERR)
        cur->fts_info = FTS_DP;
    fts_lfree(head);
    return (NULL);
}

if (sort_by_inode) {
    sp->fts_compar = fts_compare_ino;
    head = fts_sort (sp, head, nitems);
    sp->fts_compar = NULL;
}

/* Sort the entries. */
if (sp->fts_compar && nitems > 1)
    head = fts_sort(sp, head, nitems);
return (head);
}

#if GNULIB_FTS_DEBUG

struct devino {

```

```

intmax_t dev, ino;
};

#define PRINT_DEVINO "(%jd,%jd)"

static struct devino
getdevino (int fd)
{
struct stat st;
return (fd == AT_FDCWD
    ? (struct devino) { -1, 0 }
    : fstat (fd, &st) == 0
    ? (struct devino) { st.st_dev, st.st_ino }
    : (struct devino) { -1, errno });
}

/* Walk ->fts_parent links starting at E_CURR, until the root of the
current hierarchy. There should be a directory with dev/inode
matching those of AD. If not, print a lot of diagnostics. */
static void
find_matching_ancestor (FTSENT const *e_curr, struct Active_dir const *ad)
{
FTSENT const *ent;
for (ent = e_curr; ent->fts_level >= FTS_ROOTLEVEL; ent = ent->fts_parent)
{
if (ad->ino == ent->fts_statp->st_ino
    && ad->dev == ent->fts_statp->st_dev)
    return;
}
printf ("ERROR: tree dir, %s, not active\n", ad->fts_ent->fts_accpath);
printf ("active dirs:\n");
for (ent = e_curr;
     ent->fts_level >= FTS_ROOTLEVEL; ent = ent->fts_parent)
    printf ("%s%"PRIuMAX"/%"PRIuMAX") to %s%"PRIuMAX"/%"PRIuMAX"...\\n",
            ad->fts_ent->fts_accpath,
            (uintmax_t) ad->dev,
            (uintmax_t) ad->ino,
            ent->fts_accpath,
            (uintmax_t) ent->fts_statp->st_dev,
            (uintmax_t) ent->fts_statp->st_ino);
}

void
fts_cross_check (FTS const *sp)
{
FTSENT const *ent = sp->fts_cur;
FTSENT const *t;
if (!ISSET (FTS_TIGHT_CYCLE_CHECK))
    return;

Dprintf (("fts-cross-check cur=%s\\n", ent->fts_path));
/* Make sure every parent dir is in the tree. */
for (t = ent->fts_parent; t->fts_level >= FTS_ROOTLEVEL; t = t->fts_parent)
{
    struct Active_dir ad;
    ad.ino = t->fts_statp->st_ino;
    ad.dev = t->fts_statp->st_dev;
    if (!hash_lookup (sp->fts_cycle.ht, &ad))
        printf ("ERROR: active dir, %s, not in tree\\n", t->fts_path);
}
}

```

```

/* Make sure every dir in the tree is an active dir.
   But ENT is not necessarily a directory. If so, just skip this part. */
if (ent->fts_parent->fts_level >= FTS_ROOTLEVEL
    && (ent->fts_info == FTS_DP
         || ent->fts_info == FTS_D))
{
    struct Active_dir *ad;
    for (ad = hash_get_first (sp->fts_cycle.ht); ad != NULL;
         ad = hash_get_next (sp->fts_cycle.ht, ad))
    {
        find_matching_ancestor (ent, ad);
    }
}
}

static bool
same_fd (int fd1, int fd2)
{
    struct stat sb1, sb2;
    return (fstat (fd1, &sb1) == 0
            && fstat (fd2, &sb2) == 0
            && psame_inode (&sb1, &sb2));
}

static void
fd_ring_print (FTS const *sp, FILE *stream, char const *msg)
{
    if (!fts_debug)
        return;
    I_ring const *fd_ring = &sp->fts_fd_ring;
    unsigned int i = fd_ring->ir_front;
    struct devino cwd = getdevino (sp->fts_cwd_fd);
    fprintf (stream, "==== %s ===== PRINT_DEVINO\n", msg, cwd.dev, cwd.ino);
    if (i_ring_empty (fd_ring))
        return;

    while (true)
    {
        int fd = fd_ring->ir_data[i];
        if (fd < 0)
            fprintf (stream, "%u: %d:\n", i, fd);
        else
        {
            struct devino wd = getdevino (fd);
            fprintf (stream, "%u: %d: PRINT_DEVINO\n", i, fd, wd.dev, wd.ino);
        }
        if (i == fd_ring->ir_back)
            break;
        i = (i + I_RING_SIZE - 1) % I_RING_SIZE;
    }
}

/* Ensure that each file descriptor on the fd_ring matches a
parent, grandparent, etc. of the current working directory. */
static void
fd_ring_check (FTS const *sp)
{
    if (!fts_debug)
        return;

```

```

/* Make a writable copy. */
i_ring fd_w = sp->fts_fd_ring;

int cwd_fd = sp->fts_cwd_fd;
cwd_fd = fcntl (cwd_fd, F_DUPFD_CLOEXEC, STDERR_FILENO + 1);
struct devino dot = getdevino (cwd_fd);
fprintf (stderr, "===== check ===== cwd: \"PRINT_DEVINO\"\n",
         dot.dev, dot.ino);
while (! i_ring_empty (&fd_w))
{
    int fd = i_ring_pop (&fd_w);
    if (0 <= fd)
    {
        int open_flags = O_SEARCH | O_CLOEXEC;
        int parent_fd = openat (cwd_fd, "..", open_flags);
        if (parent_fd < 0)
        {
            // Warn?
            break;
        }
        if (!same_fd (fd, parent_fd))
        {
            struct devino cwd = getdevino (fd);
            fprintf (stderr, "ring : \"PRINT_DEVINO\"\n", cwd.dev, cwd.ino);
            struct devino c2 = getdevino (parent_fd);
            fprintf (stderr, "parent: \"PRINT_DEVINO\"\n", c2.dev, c2.ino);
            fts_assert (0);
        }
        close (cwd_fd);
        cwd_fd = parent_fd;
    }
}
close (cwd_fd);
#endif

static unsigned short int
internal_function
fts_stat(FTS *sp, register FTSENT *p, bool follow)
{
    struct stat *sbp = p->fts_statp;

    if (ISSET (FTS_LOGICAL)
        || (ISSET (FTS_COMFOLLOW) && p->fts_level == FTS_ROOTLEVEL))
        follow = true;

    /*
     * If doing a logical walk, or application requested FTS_FOLLOW, do
     * a stat(2). If that fails, check for a nonexistent symlink. If
     * fail, set the errno from the stat call.
     */
    int flags = follow ? 0 : AT_SYMLINK_NOFOLLOW;
    if (fstatat (sp->fts_cwd_fd, p->fts_accpath, sbp, flags) < 0)
    {
        if (follow && errno == ENOENT
            && 0 <= fstatat (sp->fts_cwd_fd, p->fts_accpath, sbp,
                             AT_SYMLINK_NOFOLLOW))
        {
            __set_errno (0);
            return FTS_SLNONE;
        }
    }
}
```

```

    }

    p->fts_errno = errno;
    memset (sbp, 0, sizeof *sbp);
    return FTS_NS;
}

if (S_ISDIR(sbp->st_mode)) {
    if (ISDOT(p->fts_name)) {
        /* Command-line "." and ".." are real directories. */
        return (p->fts_level == FTS_ROOTLEVEL ? FTS_D : FTS_DOT);
    }

    return (FTS_D);
}
if (S_ISLNK(sbp->st_mode))
    return (FTS_SL);
if (S_ISREG(sbp->st_mode))
    return (FTS_F);
return (FTS_DEFAULT);
}

static int
fts_compar (void const *a, void const *b)
{
/* Convert A and B to the correct types, to pacify the compiler, and
   for portability to bizarre hosts where "void const *" and "FTSENT
   const **" differ in runtime representation. The comparison
   function cannot modify *a and *b, but there is no compile-time
   check for this. */
FTSENT const **pa = (FTSENT const **) a;
FTSENT const **pb = (FTSENT const **) b;
return pa[0]->fts_fts->fts_compar (pa, pb);
}

static FTSENT *
internal_function
fts_sort (FTS *sp, FTSENT *head, register size_t nitems)
{
    register FTSENT **ap, *p;

/* On most modern hosts, void * and FTSENT ** have the same
run-time representation, and one can convert sp->fts_compar to
the type qsort expects without problem. Use the heuristic that
this is OK if the two pointer types are the same size, and if
converting FTSENT ** to long int is the same as converting
FTSENT ** to void * and then to long int. This heuristic isn't
valid in general but we don't know of any counterexamples. */
    FTSENT *dummy;
    int (*compare) (void const *, void const *) =
    ((sizeof &dummy == sizeof (void *))
     && (long int) &dummy == (long int) (void *) &dummy)
     ? (int (*) (void const *, void const *)) sp->fts_compar
     : fts_compar);

/*
 * Construct an array of pointers to the structures and call qsort(3).
 * Reassemble the array in the order returned by qsort. If unable to
 * sort for memory reasons, return the directory entries in their
 * current order. Allocate enough space for the current needs plus

```

```

* 40 so don't realloc one entry at a time.
*/
if (nitems > sp->fts_nitems) {
    FTSENT **a;

    sp->fts_nitems = nitems + 40;
    if (SIZE_MAX / sizeof *a < sp->fts_nitems
        || !(a = realloc (sp->fts_array,
                          sp->fts_nitems * sizeof *a))) {
        free(sp->fts_array);
        sp->fts_array = NULL;
        sp->fts_nitems = 0;
        return (head);
    }
    sp->fts_array = a;
}
for (ap = sp->fts_array, p = head; p; p = p->fts_link)
    *ap++ = p;
qsort((void *)sp->fts_array, nitems, sizeof(FTSENT *), compare);
for (head = *(ap = sp->fts_array); --nitems; ++ap)
    ap[0]->fts_link = ap[1];
ap[0]->fts_link = NULL;
return (head);
}

static FTSENT *
internal_function
fts_alloc (FTS *sp, const char *name, register size_t namelen)
{
    register FTSENT *p;
    size_t len;

    /*
     * The file name is a variable length array. Allocate the FTSENT
     * structure and the file name in one chunk.
     */
    len = FLEXSIZEOF(FTSENT, fts_name, namelen + 1);
    if ((p = malloc(len)) == NULL)
        return (NULL);

    /* Copy the name and guarantee NUL termination.*/
    memcpy(p->fts_name, name, namelen);
    p->fts_name[namelen] = '\0';

    p->fts_namelen = namelen;
    p->fts_fts = sp;
    p->fts_path = sp->fts_path;
    p->fts_errno = 0;
    p->fts_dirp = NULL;
    p->fts_flags = 0;
    p->fts_instr = FTS_NOINSTR;
    p->fts_number = 0;
    p->fts_pointer = NULL;
    return (p);
}

static void
internal_function
fts_ifree (register FTSENT *head)
{

```

```

register FTSENT *p;
int err = errno;

/* Free a linked list of structures. */
while ((p = head)) {
    head = head->fts_link;
    if (p->fts_dirp)
        closedir (p->fts_dirp);
    free(p);
}

__set_errno (err);
}

/*
* Allow essentially unlimited file name lengths; find, rm, ls should
* all work on any tree. Most systems will allow creation of file
* names much longer than MAXPATHLEN, even though the kernel won't
* resolve them. Add the size (not just what's needed) plus 256 bytes
* so don't realloc the file name 2 bytes at a time.
*/
static bool
internal_function
fts_palloc (FTS *sp, size_t more)
{
    char *p;
    size_t new_len = sp->fts_pathlen + more + 256;

    /*
     * See if fts_pathlen would overflow.
     */
    if (new_len < sp->fts_pathlen) {
        free(sp->fts_path);
        sp->fts_path = NULL;
        __set_errno (ENAMETOOLONG);
        return false;
    }
    sp->fts_pathlen = new_len;
    p = realloc(sp->fts_path, sp->fts_pathlen);
    if (p == NULL) {
        free(sp->fts_path);
        sp->fts_path = NULL;
        return false;
    }
    sp->fts_path = p;
    return true;
}

/*
* When the file name is realloc'd, have to fix all of the pointers in
* structures already returned.
*/
static void
internal_function
fts_padjust (FTS *sp, FTSENT *head)
{
    FTSENT *p;
    char *addr = sp->fts_path;

    /* This code looks at bit-patterns of freed pointers to

```

relocate them, so it relies on undefined behavior. If this trick does not work on your platform, please report a bug. \*/

```
#define ADJUST(p) do {                                \
    uintptr_t old_accpath = (uintptr_t) (p)->fts_accpath;   \
    if (old_accpath != (uintptr_t) (p)->fts_name) {        \
        (p)->fts_accpath =                               \
            addr + (old_accpath - (uintptr_t) (p)->fts_path); \
    }                                                       \
    (p)->fts_path = addr;                                \
} while (0)
/* Adjust the current set of children.*/
for (p = sp->fts_child; p; p = p->fts_link)
    ADJUST(p);

/* Adjust the rest of the tree, including the current level.*/
for (p = head; p->fts_level >= FTS_ROOTLEVEL;) {
    ADJUST(p);
    p = p->fts_link ? p->fts_link : p->fts_parent;
}
}

static size_t
internal_function _GL_ATTRIBUTE_PURE
fts_maxarglen (char * const *argv)
{
    size_t len, max;

    for (max = 0; *argv; ++argv)
        if ((len = strlen(*argv)) > max)
            max = len;
    return (max + 1);
}

/*
 * Change to dir specified by fd or file name without getting
 * tricked by someone changing the world out from underneath us.
 * Assumes p->fts_statp->st_dev and p->fts_statp->st_ino are filled in.
 * If FD is non-negative, expect it to be used after this function returns,
 * and to be closed eventually. So don't pass e.g., 'dirfd(dirp)' and then
 * do closedir(dirp), because that would invalidate the saved FD.
 * Upon failure, close FD immediately and return nonzero.
 */
static int
internal_function
fts_safe_changefd (FTS *sp, FTSENT *p, int fd, char const *dir)
{
    int ret;
    bool is_dotdot = dir && STREQ (dir, "..");
    int newfd;

    /* This clause handles the unusual case in which FTS_NOCHDIR
     * is specified, along with FTS_CWDFD. In that case, there is
     * no need to change even the virtual cwd file descriptor.
     * However, if FD is non-negative, we do close it here. */
    if (ISSET (FTS_NOCHDIR))
    {
        if (ISSET (FTS_CWDFD) && 0 <= fd)
            close (fd);
        return 0;
    }
```

```

}

if (fd < 0 && is_dotdot && ISSET (FTS_CWDFD))
{
    /* When possible, skip the diropen and subsequent fstat+dev/ino
     * comparison. I.e., when changing to parent directory
     * (chdir ("..")), use a file descriptor from the ring and
     * save the overhead of diropen+fstat, as well as avoiding
     * failure when we lack "x" access to the virtual cwd. */
    if (! i_ring_empty (&sp->fts_fd_ring))
    {
        int parent_fd;
        fd_ring_print (sp, stderr, "pre-pop");
        parent_fd = i_ring_pop (&sp->fts_fd_ring);
        if (0 <= parent_fd)
        {
            fd = parent_fd;
            dir = NULL;
        }
    }
}

newfd = fd;
if (fd < 0 && (newfd = diropen (sp, dir)) < 0)
return -1;

/* The following dev/inode check is necessary if we're doing a
 * "logical" traversal (through symlinks, a la chown -L), if the
 * system lacks O_NOFOLLOW support, or if we're changing to ".."
 * (but not via a popped file descriptor). When changing to the
 * name "..", O_NOFOLLOW can't help. In general, when the target is
 * not "..", diropen's use of O_NOFOLLOW ensures we don't mistakenly
 * follow a symlink, so we can avoid the expense of this fstat. */
if (ISSET(FTS_LOGICAL) || ! HAVE_WORKING_O_NOFOLLOW
    || (dir && STREQ (dir, "..")))
{
    struct stat sb;
    if (fstat(newfd, &sb))
    {
        ret = -1;
        goto bail;
    }
    if (p->fts_statp->st_dev != sb.st_dev
        || p->fts_statp->st_ino != sb.st_ino)
    {
        __set_errno (ENOENT);      /* disinformation */
        ret = -1;
        goto bail;
    }
}

if (ISSET(FTS_CWDFD))
{
    cwd_advance_fd (sp, newfd, ! is_dotdot);
    return 0;
}

ret = fchdir(newfd);

bail:
if (fd < 0)

```

```

    {
        int oerrno = errno;
        (void)close(newfd);
        __set_errno (oerrno);
    }
    return ret;
}
""",
"error_category": "resource management issue",
"error": "Too many open files",
"correct_code":
"""
/* Traverse a file hierarchy.

```

Copyright (C) 2004-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>. \*/

```

/*
* Copyright (c) 1990, 1993, 1994
*   The Regents of the University of California. All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*   1. Redistributions of source code must retain the above copyright
*      notice, this list of conditions and the following disclaimer.
*   2. Redistributions in binary form must reproduce the above copyright
*      notice, this list of conditions and the following disclaimer in the
*      documentation and/or other materials provided with the distribution.
*   4. Neither the name of the University nor the names of its contributors
*      may be used to endorse or promote products derived from this software
*      without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE
LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY

```

```

* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*/



#include <config.h>

#if defined LIBC_SCCS && !defined GCC_LINT && !defined lint
static char sccsid[] = "@(#)fts.c      8.6 (Berkeley) 8/14/94";
#endif


#include "fts_.h"

#if HAVE_SYS_PARAM_H || defined _LIBC
# include <sys/param.h>
#endif
#ifndef _LIBC
# include <include/sys/stat.h>
#else
# include <sys/stat.h>
#endif
#include <fcntl.h>
#include <errno.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#if !_LIBC
# include "attribute.h"
# include "fcntl--.h"
# include "flexmember.h"
# include "openat.h"
# include "opendirat.h"
# include "same-inode.h"
#endif

#include <dirent.h>
#ifndef _D_EXACT_NAMLEN
# define _D_EXACT_NAMLEN(dirent) strlen ((dirent)->d_name)
#endif

#if HAVE_STRUCT_DIRENT_D_TYPE
/* True if the type of the directory entry D is known. */
# define DT_IS_KNOWN(d) ((d)->d_type != DT_UNKNOWN)
/* True if the type of the directory entry D must be T. */
# define DT_MUST_BE(d, t) ((d)->d_type == (t))
# define D_TYPE(d) ((d)->d_type)
#else
# define DT_IS_KNOWN(d) false
# define DT_MUST_BE(d, t) false
# define D_TYPE(d) DT_UNKNOWN
#endif

# undef DT_UNKNOWN
# define DT_UNKNOWN 0

/* Any nonzero values will do here, so long as they're distinct.
Undef any existing macros out of the way. */
# undef DT_BLK
# undef DT_CHR

```

```

# undef DT_DIR
# undef DT_FIFO
# undef DT_LNK
# undef DT_REG
# undef DT_SOCK
# define DT_BLK 1
# define DT_CHR 2
# define DT_DIR 3
# define DT_FIFO 4
# define DT_LNK 5
# define DT_REG 6
# define DT_SOCK 7
#endif

#ifndef S_IFBLK
#define S_IFBLK 0
#endif
#ifndef S_IFLNK
#define S_IFLNK 0
#endif
#ifndef S_IFSOCK
#define S_IFSOCK 0
#endif

enum
{
NOT_AN_INODE_NUMBER = 0
};

#ifndef D_INO_IN_DIRENT
#define D_INO(dp) (dp)->d_ino
#else
/* Some systems don't have inodes, so fake them to avoid lots of ifdefs. */
#define D_INO(dp) NOT_AN_INODE_NUMBER
#endif

/* If possible (see max_entries, below), read no more than this many directory
entries at a time. Without this limit (i.e., when using non-NULL
fts_compar), processing a directory with 4,000,000 entries requires ~1GiB
of memory, and handling 64M entries would require 16GiB of memory. */
#ifndef FTS_MAX_READDIR_ENTRIES
#define FTS_MAX_READDIR_ENTRIES 100000
#endif

/* If there are more than this many entries in a directory,
and the conditions mentioned below are satisfied, then sort
the entries on inode number before any further processing. */
#ifndef FTS_INODE_SORT_DIR_ENTRIES_THRESHOLD
#define FTS_INODE_SORT_DIR_ENTRIES_THRESHOLD 10000
#endif

enum
{
_FTS_INODE_SORT_DIR_ENTRIES_THRESHOLD =
FTS_INODE_SORT_DIR_ENTRIES_THRESHOLD
};

enum Fts_stat
{
FTS_NO_STAT_REQUIRED = 1,

```

```

FTS_STAT_REQUIRED = 2
};

#ifndef _LIBC
# undef close
# define close __close
# undef closedir
# define closedir __closedir
# undef fchdir
# define fchdir __fchdir
# undef open
# define open __open
# undef readdir
# define readdir __readdir
#else
# undef internal_function
# define internal_function /* empty */
#endif

#ifndef __set_errno
# define __set_errno(Val) errno = (Val)
#endif

/* If this host provides the openat function, then we can avoid
attempting to open "." in some initialization code below. */
#ifndef HAVE_OPENAT
# define HAVE_OPENAT_SUPPORT 1
#else
# define HAVE_OPENAT_SUPPORT 0
#endif

#ifndef NDEBUG
# define fts_assert(expr) ((void) (0 && (expr)))
#else
# define fts_assert(expr)      \
    do {                      \
        if (!(expr))          \
            abort();           \
    } while (false)
#endif

#ifndef _LIBC
# if __glIBC_has_attribute (__fallthrough__)
# define FALLTHROUGH __attribute__ ((__fallthrough__))
# else
# define FALLTHROUGH ((void) 0)
# endif
#endif

static FTSENT *fts_alloc (FTS *, const char *, size_t) internal_function;
static FTSENT *fts_build (FTS *, int) internal_function;
static void fts_lfree (FTSENT *) internal_function;
static void fts_load (FTS *, FTSENT *) internal_function;
static size_t fts_maxarglen (char * const *) internal_function;
static void fts_padjust (FTS *, FTSENT *) internal_function;
static bool fts_palloc (FTS *, size_t) internal_function;
static FTSENT *fts_sort (FTS *, FTSENT *, size_t) internal_function;
static unsigned short int fts_stat (FTS *, FTSENT *, bool) internal_function;

```

```

static int fts_safe_changedir (FTS *, FTSENT *, int, const char *)
    internal_function;

#include "fts-cycle.c"

#ifndef MAX
#define MAX(a,b) ((a) > (b) ? (a) : (b))
#endif

#ifndef SIZE_MAX
#define SIZE_MAX ((size_t) -1)
#endif

#define ISDOT(a) (a[0] == '.' && (!a[1] || (a[1] == '.' && !a[2])))
#define STREQ(a, b) (strcmp (a, b) == 0)

#define CLR(opt) (sp->fts_options &= ~(opt))
#define ISSET(opt) ((sp->fts_options & (opt)) != 0)
#define SET(opt) (sp->fts_options |= (opt))

/* FIXME: FTS_NOCHDIR is now misnamed.
Call it FTS_USE_FULL_RELATIVE_FILE_NAMES instead. */
#define FCHDIR(sp, fd) \
(!ISSET(FTS_NOCHDIR) && (ISSET(FTS_CWDFD) \
? (cwd_advance_fd ((sp), (fd), true), 0) \
: fchdir (fd)))

/* fts_build flags */
/* FIXME: make this an enum */
#define BCHILD 1 /* fts_children */
#define BNAMES 2 /* fts_children, names only */
#define BREAD 3 /* fts_read */

#if GNULIB_FTS_DEBUG
# include <inttypes.h>
# include <stdio.h>
bool fts_debug = false;
#define Dprintf(x) do { if (fts_debug) printf x; } while (false)
static void fd_ring_check (FTS const *);
static void fd_ring_print (FTS const *, FILE *, char const *);
#else
#define Dprintf(x)
#define fd_ring_check(x)
#define fd_ring_print(a, b, c)
#endif

#define LEAVE_DIR(Fts, Ent, Tag) \
do \
{ \
    Dprintf (" %s-leaving: %s\n", Tag, (Ent)->fts_path); \
    leave_dir (Fts, Ent); \
    fd_ring_check (Fts); \
} \
while (false)

static void
fd_ring_clear (I_ring *fd_ring)
{
while ( ! i_ring_empty (fd_ring))

```

```

    {
        int fd = i_ring_pop (fd_ring);
        if (0 <= fd)
            close (fd);
    }
}

/* Overload the fts_statp->st_size member (otherwise unused, when
   fts_info is FTS_NSOK) to indicate whether fts_read should stat
   this entry or not. */
static void
fts_set_stat_required (FTSENT *p, bool required)
{
    fts_assert (p->fts_info == FTS_NSOK);
    p->fts_statp->st_size = (required
        ? FTS_STAT_REQUIRED
        : FTS_NO_STAT_REQUIRED);
}

/* Virtual fchdir. Advance SP's working directory file descriptor,
   SP->fts_cwd_fd, to FD, and push the previous value onto the fd_ring.
   CHDIR_DOWN_ONE is true if FD corresponds to an entry in the directory
   open on sp->fts_cwd_fd; i.e., to move the working directory one level
   down. */
static void
internal_function
cwd_advance_fd (FTS *sp, int fd, bool chdir_down_one)
{
    int old = sp->fts_cwd_fd;
    fts_assert (old != fd || old == AT_FDCWD);

    if (chdir_down_one)
    {
        /* Push "old" onto the ring.
           If the displaced file descriptor is non-negative, close it. */
        int prev_fd_in_slot = i_ring_push (&sp->fts_fd_ring, old);
        fd_ring_print (sp, stderr, "post-push");
        if (0 <= prev_fd_in_slot)
            close (prev_fd_in_slot); /* ignore any close failure */
    }
    else if (!ISSET (FTS_NOCHDIR))
    {
        if (0 <= old)
            close (old); /* ignore any close failure */
    }

    sp->fts_cwd_fd = fd;
}

/* Restore the initial, pre-traversal, "working directory".
   In FTS_CWDFD mode, we merely call cwd_advance_fd, otherwise,
   we may actually change the working directory.
   Return 0 upon success. Upon failure, set errno and return nonzero. */
static int
restore_initial_cwd (FTS *sp)
{
    int fail = FCHDIR (sp, ISSET (FTS_CWDFD) ? AT_FDCWD : sp->fts_rfd);
    fd_ring_clear (&(sp->fts_fd_ring));
    return fail;
}

```

```

/* Open the directory DIR if possible, and return a file
descriptor. Return -1 and set errno on failure. It doesn't matter
whether the file descriptor has read or write access. */

static int
internal_function
diopen (FTS const *sp, char const *dir)
{
    int open_flags = (O_SEARCH | O_CLOEXEC | O_DIRECTORY | O_NOCTTY | O_NONBLOCK
                     | (ISSET (FTS_PHYSICAL) ? O_NOFOLLOW : 0));

    int fd = (ISSET (FTS_CWDFD)
              ? openat (sp->fts_cwd_fd, dir, open_flags)
              : open (dir, open_flags));
    return fd;
}

FTS *
fts_open (char * const *argv,
          register int options,
          int (*compar) (FTSENT const **, FTSENT const **))
{
    register FTS *sp;
    register FTSENT *p, *root;
    register size_t nitems;
    FTSENT *parent = NULL;
    FTSENT *tmp = NULL; /* pacify gcc */
    bool defer_stat;

    /* Options check. */
    if (options & ~FTS_OPTIONMASK) {
        __set_errno (EINVAL);
        return (NULL);
    }
    if ((options & FTS_NOCHDIR) && (options & FTS_CWDFD)) {
        __set_errno (EINVAL);
        return (NULL);
    }
    if (! (options & (FTS_LOGICAL | FTS_PHYSICAL))) {
        __set_errno (EINVAL);
        return (NULL);
    }

    /* Allocate/initialize the stream */
    sp = calloc (1, sizeof *sp);
    if (sp == NULL)
        return (NULL);
    sp->fts_compar = compar;
    sp->fts_options = options;

    /* Logical walks turn on NOCHDIR; symbolic links are too hard. */
    if (ISSET(FTS_LOGICAL)) {
        SET(FTS_NOCHDIR);
        CLR(FTS_CWDFD);
    }

    /* Initialize fts_cwd_fd. */
    sp->fts_cwd_fd = AT_FDCWD;
}

```

```

if ( ISSET(FTS_CWDFD) && ! HAVE_OPENAT_SUPPORT)
{
    /* While it isn't technically necessary to open "." this
       early, doing it here saves us the trouble of ensuring
       later (where it'd be messier) that "." can in fact
       be opened. If not, revert to FTS_NOCHDIR mode. */
    int fd = open(".", O_SEARCH | O_CLOEXEC);
    if (fd < 0)
    {
        /* Even if "." is unreadable, don't revert to FTS_NOCHDIR mode
           on systems like Linux+PROC_FS, where our openat emulation
           is good enough. Note: on a system that emulates
           openat via /proc, this technique can still fail, but
           only in extreme conditions, e.g., when the working
           directory cannot be saved (i.e. save_cwd fails) --
           and that happens on Linux only when "." is unreadable
           and the CWD would be longer than PATH_MAX.
           FIXME: once Linux kernel openat support is well established,
           replace the above open call and this entire if/else block
           with the body of the if-block below. */
        if (openat_needs_fchdir())
        {
            SET(FTS_NOCHDIR);
            CLR(FTS_CWDFD);
        }
    }
    else
    {
        close(fd);
    }
}

/*
 * Start out with 1K of file name space, and enough, in any case,
 * to hold the user's file names.
*/
#ifndef MAXPATHLEN
#define MAXPATHLEN 1024
#endif
{
size_t maxarglen = fts_maxarglen(argv);
if (!fts_palloc(sp, MAX(maxarglen, MAXPATHLEN)))
    goto mem1;
}

/* Allocate/initialize root's parent. */
if (*argv != NULL) {
    if ((parent = fts_alloc(sp, "", 0)) == NULL)
        goto mem2;
    parent->fts_level = FTS_ROOTPARENTLEVEL;
}

/* The classic fts implementation would call fts_stat with
   a new entry for each iteration of the loop below.
   If the comparison function is not specified or if the
   FTS_DEFER_STAT option is in effect, don't stat any entry
   in this loop. This is an attempt to minimize the interval
   between the initial stat/lstat/fstatat and the point at which
   a directory argument is first opened. This matters for any
   directory command line argument that resides on a file system

```

```

without genuine i-nodes. If you specify FTS_DEFER_STAT along
with a comparison function, that function must not access any
data via the fts_statp pointer. */
defer_stat = (compar == NULL || ISSET(FTS_DEFER_STAT));

/* Allocate/initialize root(s). */
for (root = NULL, nitems = 0; *argv != NULL; ++argv, ++nitems) {
    /* *Do* allow zero-length file names. */
    size_t len = strlen(*argv);

    if (! (options & FTS_VERBATIM))
    {
        /* If there are two or more trailing slashes, trim all but one,
        but don't change "//" to "/", and do map "///" to "/". */
        char const *v = *argv;
        if (2 < len && v[len - 1] == '/')
            while (1 < len && v[len - 2] == '/')
                --len;
    }

    if ((p = fts_alloc(sp, *argv, len)) == NULL)
        goto mem3;
    p->fts_level = FTS_ROOTLEVEL;
    p->fts_parent = parent;
    p->fts_accpath = p->fts_name;
    /* Even when defer_stat is true, be sure to stat the first
    command line argument, since fts_read (at least with
    FTS_XDEV) requires that. */
    if (defer_stat && root != NULL) {
        p->fts_info = FTS_NSOK;
        fts_set_stat_required(p, true);
    } else {
        p->fts_info = fts_stat(sp, p, false);
    }

    /*
     * If comparison routine supplied, traverse in sorted
     * order; otherwise traverse in the order specified.
     */
    if (compar) {
        p->fts_link = root;
        root = p;
    } else {
        p->fts_link = NULL;
        if (root == NULL)
            tmp = root = p;
        else {
            tmp->fts_link = p;
            tmp = p;
        }
    }
}
if (compar && nitems > 1)
    root = fts_sort(sp, root, nitems);

/*
 * Allocate a dummy pointer and make fts_read think that we've just
 * finished the node before the root(s); set p->fts_info to FTS_INIT
 * so that everything about the "current" node is ignored.
*/

```

```

if ((sp->fts_cur = fts_alloc(sp, "", 0)) == NULL)
    goto mem3;
sp->fts_cur->fts_link = root;
sp->fts_cur->fts_info = FTS_INIT;
sp->fts_cur->fts_level = 1;
if (! setup_dir (sp))
    goto mem3;

/*
 * If using chdir(2), grab a file descriptor pointing to dot to ensure
 * that we can get back here; this could be avoided for some file names,
 * but almost certainly not worth the effort. Slashes, symbolic links,
 * and ".." are all fairly nasty problems. Note, if we can't get the
 * descriptor we run anyway, just more slowly.
*/
if (!ISSET(FTS_NOCHDIR) && !ISSET(FTS_CWDFD)
    && (sp->fts_rfd = diropen (sp, ".")) < 0)
    SET(FTS_NOCHDIR);

i_ring_init (&sp->fts_fd_ring, -1);
return (sp);

mem3: fts_lfree(root);
    free(parent);
mem2: free(sp->fts_path);
mem1: free(sp);
    return (NULL);
}

static void
internal_function
fts_load (FTS *sp, register FTSENT *p)
{
    register size_t len;
    register char *cp;

    /*
     * Load the stream structure for the next traversal. Since we don't
     * actually enter the directory until after the preorder visit, set
     * the fts_accpath field specially so the chdir gets done to the right
     * place and the user can access the first node. From fts_open it's
     * known that the file name will fit.
    */
    len = p->fts_pathlen = p->fts_namelen;
    memmove(sp->fts_path, p->fts_name, len + 1);
    if ((cp = strrchr(p->fts_name, '/')) && (cp != p->fts_name || cp[1])) {
        len = strlen(++cp);
        memmove(p->fts_name, cp, len + 1);
        p->fts_namelen = len;
    }
    p->fts_accpath = p->fts_path = sp->fts_path;
}

int
fts_close (FTS *sp)
{
    register FTSENT *freep, *p;
    int saved_errno = 0;

    /*

```

```

* This still works if we haven't read anything -- the dummy structure
* points to the root list, so we step through to the end of the root
* list which has a valid parent pointer.
*/
if (sp->fts_cur) {
    for (p = sp->fts_cur; p->fts_level >= FTS_ROOTLEVEL;) {
        freep = p;
        p = p->fts_link != NULL ? p->fts_link : p->fts_parent;
        free(freep);
    }
    free(p);
}

/* Free up child linked list, sort array, file name buffer.*/
if (sp->fts_child)
    fts_lfree(sp->fts_child);
free(sp->fts_array);
free(sp->fts_path);

if (ISSET(FTS_CWDFD))
{
    if (0 <= sp->fts_cwd_fd)
        if (close (sp->fts_cwd_fd))
            saved_errno = errno;
}
else if (!ISSET(FTS_NOCHDIR))
{
    /* Return to original directory, save errno if necessary.*/
    if (fchdir(sp->fts_rfd))
        saved_errno = errno;

    /* If close fails, record errno only if saved_errno is zero,
       so that we report the probably-more-meaningful fchdir errno. */
    if (close (sp->fts_rfd))
        if (saved_errno == 0)
            saved_errno = errno;
}

fd_ring_clear (&sp->fts_fd_ring);

if (sp->fts_leaf_optimization_works_ht)
    hash_free (sp->fts_leaf_optimization_works_ht);

free_dir (sp);

/* Free up the stream pointer.*/
free(sp);

/* Set errno and return.*/
if (saved_errno) {
    __set_errno (saved_errno);
    return (-1);
}

return (0);
}

/* Minimum link count of a traditional Unix directory. When leaf
optimization is OK and a directory's st_nlink == MIN_DIR_NLINK,
then the directory has no subdirectories. */

```

```

enum { MIN_DIR_NLINK = 2 };

/* Whether leaf optimization is OK for a directory. */
enum leaf_optimization
{
    /* st_nlink is not reliable for this directory's subdirectories. */
    NO_LEAF_OPTIMIZATION,
    /* st_nlink == 2 means the directory lacks subdirectories. */
    OK_LEAF_OPTIMIZATION
};

#ifndef __linux__ || defined __ANDROID__
&& HAVE_SYS_VFS_H && HAVE_FSTATFS && HAVE_STRUCT_STATFS_F_TYPE
#endif

#include <sys/vfs.h>

/* Linux-specific constants from coreutils' src/fs.h */
#define S_MAGIC_AFS 0x5346414F
#define S_MAGIC_CIFS 0xFF534D42
#define S_MAGIC_NFS 0x6969
#define S_MAGIC_PROC 0x9FA0
#define S_MAGIC_TMPFS 0x1021994

#ifndef HAVE_FSWORD_T
typedef __fsword_t fsword;
#else
typedef long int fsword;
#endif

/* Map a stat.st_dev number to a file system type number f_ftype. */
struct dev_type
{
    dev_t st_dev;
    fsword f_type;
};

/* Use a tiny initial size. If a traversal encounters more than
   a few devices, the cost of growing/rehashing this table will be
   rendered negligible by the number of inodes processed. */
enum { DEV_TYPE_HT_INITIAL_SIZE = 13 };

static size_t
dev_type_hash (void const *x, size_t table_size)
{
    struct dev_type const *ax = x;
    uintmax_t dev = ax->st_dev;
    return dev % table_size;
}

static bool
dev_type_compare (void const *x, void const *y)
{
    struct dev_type const *ax = x;
    struct dev_type const *ay = y;
    return ax->st_dev == ay->st_dev;
}

/* Return the file system type of P with file descriptor FD, or 0 if not known.
   If FD is negative, P's file descriptor is unavailable.

```

```

Try to cache known values. */

static fsword
filesystem_type (FTSENT const *p, int fd)
{
    FTS *sp = p->fts_fts;
    Hash_table *h = sp->fts_leaf_optimization_works_ht;
    struct dev_type *ent;
    struct statfs fs_buf;

    /* If we're not in CWDFD mode, don't bother with this optimization,
       since the caller is not serious about performance. */
    if (!ISSET (FTS_CWDFD))
        return 0;

    if (!h)
        h = sp->fts_leaf_optimization_works_ht
        = hash_initialize (DEV_TYPE_HT_INITIAL_SIZE, NULL, dev_type_hash,
                           dev_type_compare, free);
    if (h)
    {
        struct dev_type tmp;
        tmp.st_dev = p->fts_statp->st_dev;
        ent = hash_lookup (h, &tmp);
        if (ent)
            return ent->f_type;
    }

    /* Look-up failed. Query directly and cache the result. */
    if (fd < 0 || fstatfs (fd, &fs_buf) != 0)
        return 0;

    if (h)
    {
        struct dev_type *t2 = malloc (sizeof *t2);
        if (t2)
        {
            t2->st_dev = p->fts_statp->st_dev;
            t2->f_type = fs_buf.f_type;

            ent = hash_insert (h, t2);
            if (ent)
                fts_assert (ent == t2);
            else
                free (t2);
        }
    }

    return fs_buf.f_type;
}

/* Return true if sorting dirents on inode numbers is known to improve
traversal performance for the directory P with descriptor DIR_FD.
Return false otherwise. When in doubt, return true.
DIR_FD is negative if unavailable. */
static bool
dirent_inode_sort_may_be_useful (FTSENT const *p, int dir_fd)
{
    /* Skip the sort only if we can determine efficiently
       that skipping it is the right thing to do.

```



```

return true;
}
static enum leaf_optimization
leaf_optimization (_GL_UNUSED FTSENT const *p, _GL_UNUSED int dir_fd)
{
return NO_LEAF_OPTIMIZATION;
}
#endif

/*
* Special case of "/" at the end of the file name so that slashes aren't
* appended which would cause file names to be written as "....//foo".
*/
#define NAPPEND(p) \
    (p->fts_path[p->fts_pathlen - 1] == '/' \
     ? p->fts_pathlen - 1 : p->fts_pathlen) \
                                \
FTSENT *
fts_read (register FTS *sp)
{
register FTSENT *p, *tmp;
register unsigned short int instr;
register char *t;

/* If finished or unrecoverable error, return NULL. */
if (sp->fts_cur == NULL || ISSET(FTS_STOP))
    return (NULL);

/* Set current node pointer. */
p = sp->fts_cur;

/* Save and zero out user instructions. */
instr = p->fts_instr;
p->fts_instr = FTS_NOINSTR;

/* Any type of file may be re-visited; re-stat and re-turn. */
if (instr == FTS_AGAIN) {
    p->fts_info = fts_stat(sp, p, false);
    return (p);
}
Dprintf (("fts_read: p=%s\n",
          p->fts_info == FTS_INIT ? "" : p->fts_path));

/*
* Following a symlink -- SLNONE test allows application to see
* SLNONE and recover. If indirecting through a symlink, have
* keep a pointer to current location. If unable to get that
* pointer, follow fails.
*/
if (instr == FTS_FOLLOW &&
    (p->fts_info == FTS_SL || p->fts_info == FTS_SLNONE)) {
    p->fts_info = fts_stat(sp, p, true);
    if (p->fts_info == FTS_D && !ISSET(FTS_NOCHDIR)) {
        if ((p->fts_symfd = diropen (sp, ".") < 0) {
            p->fts_errno = errno;
            p->fts_info = FTS_ERR;
        } else
            p->fts_flags |= FTS_SYMFOLLOW;
    }
    goto check_for_dir;
}

```

```

}

/* Directory in pre-order.*/
if (p->fts_info == FTS_D) {
    /* If skipped or crossed mount point, do post-order visit.*/
    if (instr == FTS_SKIP ||
        (ISSET(FTS_XDEV) && p->fts_statp->st_dev != sp->fts_dev)) {
        if (p->fts_flags & FTS_SYMFOLLOW)
            (void)close(p->fts_symfd);
        if (sp->fts_child) {
            fts_lfree(sp->fts_child);
            sp->fts_child = NULL;
        }
        p->fts_info = FTS_DP;
        LEAVE_DIR (sp, p, "1");
        return (p);
    }

/* Rebuild if only read the names and now traversing.*/
if (sp->fts_child != NULL && ISSET(FTS_NAMEONLY)) {
    CLR(FTS_NAMEONLY);
    fts_lfree(sp->fts_child);
    sp->fts_child = NULL;
}

/*
 * Cd to the subdirectory.
 *
 * If have already read and now fail to chdir, whack the list
 * to make the names come out right, and set the parent errno
 * so the application will eventually get an error condition.
 * Set the FTS_DONTCHDIR flag so that when we logically change
 * directories back to the parent we don't do a chdir.
 *
 * If haven't read do so. If the read fails, fts_build sets
 * FTS_STOP or the fts_info field of the node.
 */
if (sp->fts_child != NULL) {
    if (fts_safe_changedir(sp, p, -1, p->fts_acccpath)) {
        p->fts_errno = errno;
        p->fts_flags |= FTS_DONTCHDIR;
        for (p = sp->fts_child; p != NULL;
             p = p->fts_link)
            p->fts_acccpath =
                p->fts_parent->fts_acccpath;
    }
} else if ((sp->fts_child = fts_build(sp, BREAD)) == NULL) {
    if (ISSET(FTS_STOP))
        return (NULL);
    /* If fts_build's call to fts_safe_changedir failed
     * because it was not able to fchdir into a
     * subdirectory, tell the caller. */
    if (p->fts_errno && p->fts_info != FTS_DNR)
        p->fts_info = FTS_ERR;
    LEAVE_DIR (sp, p, "2");
    return (p);
}
p = sp->fts_child;
sp->fts_child = NULL;
goto name;

```

```
}
```

```
/* Move to the next node on this level. */
next: tmp = p;

/* If we have so many directory entries that we're reading them
in batches, and we've reached the end of the current batch,
read in a new batch. */
if (p->fts_link == NULL && p->fts_parent->fts_dirp)
{
    p = tmp->fts_parent;
    sp->fts_cur = p;
    sp->fts_path[p->fts_pathlen] = '\0';

    if ((p = fts_build (sp, BREAD)) == NULL)
    {
        if (ISSET(FTS_STOP))
            return NULL;
        goto cd_dot_dot;
    }

    free(tmp);
    goto name;
}

if ((p = p->fts_link) != NULL) {
    sp->fts_cur = p;
    free(tmp);

    /*
     * If reached the top, return to the original directory (or
     * the root of the tree), and load the file names for the next
     * root.
     */
    if (p->fts_level == FTS_ROOTLEVEL) {
        if (restore_initial_cwd(sp)) {
            SET(FTS_STOP);
            return (NULL);
        }
        free_dir(sp);
        fts_load(sp, p);
        if (! setup_dir(sp)) {
            free_dir(sp);
            return (NULL);
        }
        goto check_for_dir;
    }

    /*
     * User may have called fts_set on the node. If skipped,
     * ignore. If followed, get a file descriptor so we can
     * get back if necessary.
     */
    if (p->fts_instr == FTS_SKIP)
        goto next;
    if (p->fts_instr == FTS_FOLLOW) {
        p->fts_info = fts_stat(sp, p, true);
        if (p->fts_info == FTS_D && !ISSET(FTS_NOCHDIR)) {
            if ((p->fts_symfd = diropen (sp, ".") < 0) {
                p->fts_errno = errno;

```

```

                p->fts_info = FTS_ERR;
            } else
                p->fts_flags |= FTS_SYMFOLLOW;
        }
        p->fts_instr = FTS_NOINSTR;
    }

name:      t = sp->fts_path + NAPPEND(p->fts_parent);
*t++ = '/';
memmove(t, p->fts_name, p->fts_namelen + 1);
check_for_dir:
    sp->fts_cur = p;
    if (p->fts_info == FTS_NSOK)
    {
        if (p->fts_statp->st_size == FTS_STAT_REQUIRED)
            p->fts_info = fts_stat(sp, p, false);
        else
            fts_assert (p->fts_statp->st_size == FTS_NO_STAT_REQUIRED);
    }

    if (p->fts_info == FTS_D)
    {
        /* Now that P->fts_statp is guaranteed to be valid,
        if this is a command-line directory, record its
        device number, to be used for FTS_XDEV. */
        if (p->fts_level == FTS_ROOTLEVEL)
            sp->fts_dev = p->fts_statp->st_dev;
        Dprintf ((" entering: %s\n", p->fts_path));
        if (! enter_dir (sp, p))
            return NULL;
    }
    return p;
}
cd_dot_dot:
/* Move up to the parent node. */
p = tmp->fts_parent;
sp->fts_cur = p;
free(tmp);

if (p->fts_level == FTS_ROOTPARENTLEVEL) {
/*
 * Done; free everything up and set errno to 0 so the user
 * can distinguish between error and EOF.
 */
free(p);
__set_errno (0);
return (sp->fts_cur = NULL);
}

fts_assert (p->fts_info != FTS_NSOK);

/* NUL terminate the file name. */
sp->fts_path[p->fts_pathlen] = '\0';

/*
 * Return to the parent directory. If at a root node, restore
 * the initial working directory. If we came through a symlink,
 * go back through the file descriptor. Otherwise, move up
 * one level, via "..".

```

```

*/
if (p->fts_level == FTS_ROOTLEVEL) {
    if (restore_initial_cwd(sp)) {
        p->fts_errno = errno;
        SET(FTS_STOP);
    }
} else if (p->fts_flags & FTS_SYMFOLLOW) {
    if (FCHDIR(sp, p->fts_symfd)) {
        p->fts_errno = errno;
        SET(FTS_STOP);
    }
    (void)close(p->fts_symfd);
} else if (!(p->fts_flags & FTS_DONTCHDIR) &&
           fts_safe_changedir(sp, p->fts_parent, -1, "..")) {
    p->fts_errno = errno;
    SET(FTS_STOP);
}

/* If the directory causes a cycle, preserve the FTS_DC flag and keep
   the corresponding dev/ino pair in the hash table. It is going to be
   removed when leaving the original directory. */
if (p->fts_info != FTS_DC) {
    p->fts_info = p->fts_errno ? FTS_ERR : FTS_DP;
    if (p->fts_errno == 0)
        LEAVE_DIR (sp, p, "3");
}
return ISSET(FTS_STOP) ? NULL : p;
}

/*
* Fts_set takes the stream as an argument although it's not used in this
* implementation; it would be necessary if anyone wanted to add global
* semantics to fts using fts_set. An error return is allowed for similar
* reasons.
*/
/* ARGSUSED */
int
fts_set(_GL_UNUSED FTS *sp, FTSENT *p, int instr)
{
    if (instr != 0 && instr != FTS_AGAIN && instr != FTS_FOLLOW &&
        instr != FTS_NOINSTR && instr != FTS_SKIP) {
        __set_errno (EINVAL);
        return (1);
    }
    p->fts_instr = instr;
    return (0);
}

FTSENT *
fts_children (register FTS *sp, int instr)
{
    register FTSENT *p;
    int fd;

    if (instr != 0 && instr != FTS_NAMEONLY) {
        __set_errno (EINVAL);
        return (NULL);
    }

    /* Set current node pointer. */

```

```

p = sp->fts_cur;

/*
 * Errno set to 0 so user can distinguish empty directory from
 * an error.
 */
__set_errno (0);

/* Fatal errors stop here. */
if (ISSET(FTS_STOP))
    return (NULL);

/* Return logical hierarchy of user's arguments. */
if (p->fts_info == FTS_INIT)
    return (p->fts_link);

/*
 * If not a directory being visited in pre-order, stop here. Could
 * allow FTS_DNR, assuming the user has fixed the problem, but the
 * same effect is available with FTS AGAIN.
 */
if (p->fts_info != FTS_D /* && p->fts_info != FTS_DNR */)
    return (NULL);

/* Free up any previous child list. */
if (sp->fts_child != NULL)
    fts_lfree(sp->fts_child);

if (instr == FTS_NAMEONLY) {
    SET(FTS_NAMEONLY);
    instr = BNAMES;
} else
    instr = BCHILD;

/*
 * If using chdir on a relative file name and called BEFORE fts_read
 * does its chdir to the root of a traversal, we can lose -- we need to
 * chdir into the subdirectory, and we don't know where the current
 * directory is, so we can't get back so that the upcoming chdir by
 * fts_read will work.
 */
if (p->fts_level != FTS_ROOTLEVEL || p->fts_acccpath[0] == '/' ||
    ISSET(FTS_NOCHDIR))
    return (sp->fts_child = fts_build(sp, instr));

if ((fd = diropen (sp, ".") < 0)
    return (sp->fts_child = NULL);
sp->fts_child = fts_build(sp, instr);
if (ISSET(FTS_CWDFD))
{
    cwd_advance_fd (sp, fd, true);
}
else
{
    if (fchdir(fd))
    {
        int saved_errno = errno;
        close (fd);
        __set_errno (saved_errno);
        return NULL;
    }
}

```

```

        }
    close (fd);
}
return (sp->fts_child);
}

/* A comparison function to sort on increasing inode number.
For some file system types, sorting either way makes a huge
performance difference for a directory with very many entries,
but sorting on increasing values is slightly better than sorting
on decreasing values. The difference is in the 5% range. */
static int
fts_compare_ino (struct _ftsent const **a, struct _ftsent const **b)
{
return _GL_CMP (a[0]->fts_statp->st_ino, b[0]->fts_statp->st_ino);
}

/* Map the dirent.d_type value, DTTYPE, to the corresponding stat.st_mode
S_IF* bit and set ST.st_mode, thus clearing all other bits in that field. */
static void
set_stat_type (struct stat *st, unsigned int dtype)
{
mode_t type;
switch (dtype)
{
case DT_BLK:
type = S_IFBLK;
break;
case DT_CHR:
type = S_IFCHR;
break;
case DT_DIR:
type = S_IFDIR;
break;
case DT_FIFO:
type = S_IFIFO;
break;
case DT_LNK:
type = S_IFLNK;
break;
case DT_REG:
type = S_IFREG;
break;
case DT_SOCK:
type = S_IFSOCK;
break;
default:
type = 0;
}
st->st_mode = type;
}

#define closedir_and_clear(dirp)          \
do {                                       \
    closedir (dirp);                      \
    dirp = NULL;                          \
} while (0)

```

```

#define fts_opendir(file, Pdir_fd) \
    opendirat(! ISSET(FTS_NOCHDIR) && ISSET(FTS_CWDFD) \
        ? sp->fts_cwd_fd : AT_FDCWD, \
        file, \
        (((ISSET(FTS_PHYSICAL) \
            && ! (ISSET(FTS_COMFOLLOW) \
                && cur->fts_level == FTS_ROOTLEVEL)) \
            ? O_NOFOLLOW : 0)), \
        Pdir_fd)

/*
 * This is the tricky part -- do not casually change *anything* in here. The
 * idea is to build the linked list of entries that are used by fts_children
 * and fts_read. There are lots of special cases.
 *
 * The real slowdown in walking the tree is the stat calls. If FTS_NOSTAT is
 * set and it's a physical walk (so that symbolic links can't be directories),
 * we can do things quickly. First, if it's a 4.4BSD file system, the type
 * of the file is in the directory entry. Otherwise, we assume that the number
 * of subdirectories in a node is equal to the number of links to the parent.
 * The former skips all stat calls. The latter skips stat calls in any leaf
 * directories and for any files after the subdirectories in the directory have
 * been found, cutting the stat calls by about 2/3.
 */
static FTSENT *
internal_function
fts_build (register FTS *sp, int type)
{
    register FTSENT *p, *head;
    register size_t nitems;
    FTSENT *tail;
    int saved_errno;
    bool descend;
    bool doadjust;
    ptrdiff_t level;
    size_t len, maxlen, new_len;
    char *cp;
    int dir_fd;
    FTSENT *cur = sp->fts_cur;
    bool continue_readdir = !!cur->fts_dirp;
    bool sort_by_inode = false;
    size_t max_entries;

    /* When cur->fts_dirp is non-NULL, that means we should
     * continue calling readdir on that existing DIR* pointer
     * rather than opening a new one. */
    if (continue_readdir)
    {
        DIR *dp = cur->fts_dirp;
        dir_fd = dirfd (dp);
        if (dir_fd < 0)
        {
            int dirfd_errno = errno;
            closedir_and_clear (cur->fts_dirp);
            if (type == BREAD)
            {
                cur->fts_info = FTS_DNR;
                cur->fts_errno = dirfd_errno;
            }
            return NULL;
        }
    }
}

```

```

        }
    }
else
{
    /* Open the directory for reading. If this fails, we're done.
     * If being called from fts_read, set the fts_info field. */
    if ((cur->fts_dirp = fts_opendir(cur->fts_accpth, &dir_fd)) == NULL)
    {
        if (type == BREAD)
        {
            cur->fts_info = FTS_DNR;
            cur->fts_errno = errno;
        }
        return NULL;
    }
    /* Rather than calling fts_stat for each and every entry encountered
     * in the readdir loop (below), stat each directory only right after
     * opening it. */
    bool stat_optimization = cur->fts_info == FTS_NSOK;

    if (stat_optimization
        /* Also read the stat info again after opening a directory to
         * reveal eventual changes caused by a submount triggered by
         * the traversal. But do it only for utilities which use
         * FTS_TIGHT_CYCLE_CHECK. Therefore, only find and du
         * benefit/suffer from this feature for now. */
        || ISSET(FTS_TIGHT_CYCLE_CHECK))
    {
        if (!stat_optimization)
            LEAVE_DIR(sp, cur, "4");
        if (fstat(dir_fd, cur->fts_statp) != 0)
        {
            int fstat_errno = errno;
            closedir_and_clear(cur->fts_dirp);
            if (type == BREAD)
            {
                cur->fts_errno = fstat_errno;
                cur->fts_info = FTS_NS;
            }
            __set_errno(fstat_errno);
            return NULL;
        }
        if (stat_optimization)
            cur->fts_info = FTS_D;
        else if (!enter_dir(sp, cur))
        {
            int err = errno;
            closedir_and_clear(cur->fts_dirp);
            __set_errno(err);
            return NULL;
        }
    }
}

```

/\* Maximum number of readdir entries to read at one time. This limitation is to avoid reading millions of entries into memory at once. When an fts\_compar function is specified, we have no choice: we must read all entries into memory before calling that function. But when no such function is specified, we can read entries in batches that are large enough to help us with inode-

```

sorting, yet not so large that we risk exhausting memory. */
max_entries = sp->fts_compar ? SIZE_MAX : FTS_MAX_READDIR_ENTRIES;

/*
 * If we're going to need to stat anything or we want to descend
 * and stay in the directory, chdir. If this fails we keep going,
 * but set a flag so we don't chdir after the post-order visit.
 * We won't be able to stat anything, but we can still return the
 * names themselves. Note, that since fts_read won't be able to
 * chdir into the directory, it will have to return different file
 * names than before, i.e. "a/b" instead of "b". Since the node
 * has already been visited in pre-order, have to wait until the
 * post-order visit to return the error. There is a special case
 * here, if there was nothing to stat then it's not an error to
 * not be able to stat. This is all fairly nasty. If a program
 * needed sorted entries or stat information, they had better be
 * checking FTS_NS on the returned nodes.
*/
if (continue_readdir)
{
    /* When resuming a short readdir run, we already have
     * the required dirp and dir_fd. */
    descend = true;
}
else
{
    /* Try to descend unless it is a names-only fts_children,
     * or the directory is known to lack subdirectories. */
    descend = (type != BNAMES
        && ! (ISSET(FTS_NOSTAT) && ISSET(FTS_PHYSICAL)
            && ! ISSET(FTS_SEEDOT)
            && cur->fts_statp->st_nlink == MIN_DIR_NLINK
            && (leaf_optimization(cur, dir_fd)
                != NO_LEAF_OPTIMIZATION)));
    if (descend || type == BREAD)
    {
        if (ISSET(FTS_CWDFD)) {
            int old_fd = dir_fd;
            dir_fd = fcntl(dir_fd, F_DUPFD_CLOEXEC, STDERR_FILENO + 1);
            close(old_fd); // avoid lingering open file descriptors
        }
        if (dir_fd < 0 || fts_safe_changedir(sp, cur, dir_fd, NULL)) {
            if (descend && type == BREAD)
                cur->fts_errno = errno;
            cur->fts_flags |= FTS_DONTCHDIR;
            descend = false;
            closedir_and_clear(cur->fts_dirp);
            if (ISSET(FTS_CWDFD) && 0 <= dir_fd)
                close(dir_fd);
            cur->fts_dirp = NULL;
        } else
            descend = true;
    }
}

/*
 * Figure out the max file name length that can be stored in the
 * current buffer -- the inner loop allocates more space as necessary.
 * We really wouldn't have to do the maxlen calculations here, we
 * could do them in fts_read before returning the name, but it's a

```

```

* lot easier here since the length is part of the dirent structure.
*
* If not changing directories set a pointer so that can just append
* each new component into the file name.
*/
len = NAPPEND(cur);
if (!ISSET(FTS_NOCHDIR)) {
    cp = sp->fts_path + len;
    *cp++ = '/';
} else {
    /* GCC, you're too verbose. */
    cp = NULL;
}
len++;
maxlen = sp->fts_pathlen - len;

level = cur->fts_level + 1;

/* Read the directory, attaching each entry to the "link" pointer. */
doadjust = false;
head = NULL;
tail = NULL;
nitems = 0;
while (cur->fts_dirp) {
    size_t d_namelen;
    __set_errno (0);
    struct dirent *dp = readdir(cur->fts_dirp);
    if (dp == NULL) {
        if (errno) {
            cur->fts_errno = errno;
            /* If we've not read any items yet, treat
               the error as if we can't access the dir. */
            cur->fts_info = (continue_readdir || nitems)
                ? FTS_ERR : FTS_DNR;
        }
        closedir_and_clear(cur->fts_dirp);
        break;
    }
    if (!ISSET(FTS_SEEDOT) && ISDOT(dp->d_name))
        continue;

    d_namelen = _D_EXACT_NAMLEN (dp);
    p = fts_alloc (sp, dp->d_name, d_namelen);
    if (!p)
        goto mem1;
    if (d_namelen >= maxlen) {
        /* include space for NUL */
        uintptr_t oldaddr = (uintptr_t) sp->fts_path;
        if (! fts_palloc(sp, d_namelen + len + 1)) {
            /*
             * No more memory. Save
             * errno, free up the current structure and the
             * structures already allocated.
            */
            saved_errno = errno;
            free(p);
            fts_lfree(head);
            closedir_and_clear(cur->fts_dirp);
            cur->fts_info = FTS_ERR;
            SET(FTS_STOP);
        }
    }
mem1:

```

```

__set_errno (saved_errno);
return (NULL);
}
/* Did realloc() change the pointer? */
if (oldaddr != (uintptr_t) sp->fts_path) {
    doadjust = true;
    if (!ISSET(FTS_NOCHDIR))
        cp = sp->fts_path + len;
}
maxlen = sp->fts_pathlen - len;
}

new_len = len + d_namelen;
if (new_len < len) {
    /*
     * In the unlikely event that we would end up
     * with a file name longer than SIZE_MAX, free up
     * the current structure and the structures already
     * allocated, then error out with ENAMETOOLONG.
     */
    free(p);
    fts_lfree(head);
    closedir_and_clear(cur->fts_dirp);
    cur->fts_info = FTS_ERR;
    SET(FTS_STOP);
    __set_errno (ENAMETOOLONG);
    return (NULL);
}
p->fts_level = level;
p->fts_parent = sp->fts_cur;
p->fts_pathlen = new_len;

/* Store dirent.d_ino, in case we need to sort
entries before processing them. */
p->fts_statp->st_ino = D_INO (dp);

/* Build a file name for fts_stat to stat. */
if (!ISSET(FTS_NOCHDIR)) {
    p->fts_acccpath = p->fts_path;
    memmove(cp, p->fts_name, p->fts_namelen + 1);
} else
    p->fts_acccpath = p->fts_name;

if (sp->fts_compar == NULL || !ISSET(FTS_DEFER_STAT)) {
    /* Record what fts_read will have to do with this
entry. In many cases, it will simply fts_stat it,
but we can take advantage of any d_type information
to optimize away the unnecessary stat calls. I.e.,
if FTS_NOSTAT is in effect and we're not following
symlinks (FTS_PHYSICAL) and d_type indicates this
is *not* a directory, then we won't have to stat it
at all. If it *is* a directory, then (currently)
we stat it regardless, in order to get device and
inode numbers. Some day we might optimize that
away, too, for directories where d_ino is known to
be valid. */
    bool skip_stat = (!ISSET(FTS_NOSTAT)
                    && DT_IS_KNOWN(dp)
                    && !DT_MUST_BE(dp, DT_DIR)
                    && (!ISSET(FTS_PHYSICAL)

```

```

                || ! DT_MUST_BE(dp, DT_LNK));
p->fts_info = FTS_NSOK;
/* Propagate dirent.d_type information back
to caller, when possible. */
set_stat_type (p->fts_statp, D_TYPE (dp));
fts_set_stat_required(p, !skip_stat);
} else {
    p->fts_info = fts_stat(sp, p, false);
}

/* We walk in directory order so "ls -f" doesn't get upset.*/
p->fts_link = NULL;
if (head == NULL)
    head = tail = p;
else {
    tail->fts_link = p;
    tail = p;
}

/* If there are many entries, no sorting function has been
specified, and this file system is of a type that may be
slow with a large number of entries, arrange to sort the
directory entries on increasing inode numbers.

The NITEMS comparison uses ==, not >, because the test
needs to be tried at most once once, and NITEMS will exceed
the threshold after it is incremented below. */
if (nitems == _FTS_INODE_SORT_DIR_ENTRIES_THRESHOLD
    && !sp->fts_compar)
sort_by_inode = dirent_inode_sort_may_be_useful (cur, dir_fd);

++nitems;
if (max_entries <= nitems) {
    /* When there are too many dir entries, leave
       fts_dirp open, so that a subsequent fts_read
       can take up where we leave off. */
    break;
}
}

/*
* If realloc() changed the address of the file name, adjust the
* addresses for the rest of the tree and the dir list.
*/
if (doadjust)
    fts_padjust(sp, head);

/*
* If not changing directories, reset the file name back to original
* state.
*/
if (!SSET(FTS_NOCHDIR)) {
    if (len == sp->fts_pathlen || nitems == 0)
        --cp;
    *cp = '\0';
}

/*
* If descended after called from fts_children or after called from
* fts_read and nothing found, get back. At the root level we use

```

```

 * the saved fd; if one of fts_open()'s arguments is a relative name
 * to an empty directory, we wind up here with no other way back. If
 * can't get back, we're done.
 */
if (!continue_readdir && descend && (type == BCHILD || !nitems) &&
    (cur->fts_level == FTS_ROOTLEVEL
     ? restore_initial_cwd(sp)
     : fts_safe_changefile(sp, cur->fts_parent, -1, "..")) {
    cur->fts_info = FTS_ERR;
    SET(FTS_STOP);
    fts_lfree(head);
    return (NULL);
}

/* If didn't find anything, return NULL. */
if (!nitems) {
    if (type == BREAD
        && cur->fts_info != FTS_DNR && cur->fts_info != FTS_ERR)
        cur->fts_info = FTS_DP;
    fts_lfree(head);
    return (NULL);
}

if (sort_by_inode) {
    sp->fts_compar = fts_compare_ino;
    head = fts_sort (sp, head, nitems);
    sp->fts_compar = NULL;
}

/* Sort the entries. */
if (sp->fts_compar && nitems > 1)
    head = fts_sort(sp, head, nitems);
return (head);
}

#if GNULIB_FTS_DEBUG

struct devino {
intmax_t dev, ino;
};

#define PRINT_DEVINO "(%jd,%jd)"

static struct devino
getdevino (int fd)
{
struct stat st;
return (fd == AT_FDCWD
    ? (struct devino) { -1, 0 }
    : fstat (fd, &st) == 0
    ? (struct devino) { st.st_dev, st.st_ino }
    : (struct devino) { -1, errno });
}

/* Walk ->fts_parent links starting at E_CURR, until the root of the
current hierarchy. There should be a directory with dev/inode
matching those of AD. If not, print a lot of diagnostics. */
static void
find_matching_ancestor (FTSENT const *e_curr, struct Active_dir const *ad)
{
FTSENT const *ent;

```

```

for (ent = e_curr; ent->fts_level >= FTS_ROOTLEVEL; ent = ent->fts_parent)
{
    if (ad->ino == ent->fts_statp->st_ino
        && ad->dev == ent->fts_statp->st_dev)
        return;
}
printf ("ERROR: tree dir, %s, not active\n", ad->fts_ent->fts_accpath);
printf ("active dirs:\n");
for (ent = e_curr;
     ent->fts_level >= FTS_ROOTLEVEL; ent = ent->fts_parent)
    printf (" %s%"PRIuMAX"/%"PRIuMAX") to %s(%"PRIuMAX"/%"PRIuMAX")...\n",
            ad->fts_ent->fts_accpath,
            (uintmax_t) ad->dev,
            (uintmax_t) ad->ino,
            ent->fts_accpath,
            (uintmax_t) ent->fts_statp->st_dev,
            (uintmax_t) ent->fts_statp->st_ino);
}

void
fts_cross_check (FTS const *sp)
{
FTSENT const *ent = sp->fts_cur;
FTSENT const *t;
if (!ISSET (FTS_TIGHT_CYCLE_CHECK))
    return;

Dprintf (("fts-cross-check cur=%s\n", ent->fts_path));
/* Make sure every parent dir is in the tree. */
for (t = ent->fts_parent; t->fts_level >= FTS_ROOTLEVEL; t = t->fts_parent)
{
    struct Active_dir ad;
    ad.ino = t->fts_statp->st_ino;
    ad.dev = t->fts_statp->st_dev;
    if (!hash_lookup (sp->fts_cycle.ht, &ad))
        printf ("ERROR: active dir, %s, not in tree\n", t->fts_path);
}

/* Make sure every dir in the tree is an active dir.
   But ENT is not necessarily a directory. If so, just skip this part. */
if (ent->fts_parent->fts_level >= FTS_ROOTLEVEL
    && (ent->fts_info == FTS_DP
          || ent->fts_info == FTS_D))
{
    struct Active_dir *ad;
    for (ad = hash_get_first (sp->fts_cycle.ht); ad != NULL;
         ad = hash_get_next (sp->fts_cycle.ht, ad))
    {
        find_matching_ancestor (ent, ad);
    }
}

static bool
same_fd (int fd1, int fd2)
{
struct stat sb1, sb2;
return (fstat (fd1, &sb1) == 0
        && fstat (fd2, &sb2) == 0
        && psame_inode (&sb1, &sb2));
}

```

```

}

static void
fd_ring_print (FTS const *sp, FILE *stream, char const *msg)
{
if (!fts_debug)
    return;
I_ring const *fd_ring = &sp->fts_fd_ring;
unsigned int i = fd_ring->ir_front;
struct devino cwd = getdevino (sp->fts_cwd_fd);
fprintf (stream, "==== %s ===== PRINT_DEVINO\n", msg, cwd.dev, cwd.ino);
if (i_ring_empty (fd_ring))
    return;

while (true)
{
    int fd = fd_ring->ir_data[i];
    if (fd < 0)
        fprintf (stream, "%u: %d:\n", i, fd);
    else
    {
        struct devino wd = getdevino (fd);
        fprintf (stream, "%u: %d: PRINT_DEVINO\n", i, fd, wd.dev, wd.ino);
    }
    if (i == fd_ring->ir_back)
        break;
    i = (i + I_RING_SIZE - 1) % I_RING_SIZE;
}
}

/* Ensure that each file descriptor on the fd_ring matches a
parent, grandparent, etc. of the current working directory. */
static void
fd_ring_check (FTS const *sp)
{
if (!fts_debug)
    return;

/* Make a writable copy. */
I_ring fd_w = sp->fts_fd_ring;

int cwd_fd = sp->fts_cwd_fd;
cwd_fd = fcntl (cwd_fd, F_DUPFD_CLOEXEC, STDERR_FILENO + 1);
struct devino dot = getdevino (cwd_fd);
fprintf (stderr, "===== check ===== cwd: PRINT_DEVINO\n",
        dot.dev, dot.ino);
while ( ! i_ring_empty (&fd_w))
{
    int fd = i_ring_pop (&fd_w);
    if (0 <= fd)
    {
        int open_flags = O_SEARCH | O_CLOEXEC;
        int parent_fd = openat (cwd_fd, "..", open_flags);
        if (parent_fd < 0)
        {
            // Warn?
            break;
        }
        if (!same_fd (fd, parent_fd))
        {

```

```

        struct devino cwd = getdevino (fd);
        fprintf (stderr, "ring : PRINT_DEVINO\n", cwd.dev, cwd.ino);
        struct devino c2 = getdevino (parent_fd);
        fprintf (stderr, "parent: PRINT_DEVINO\n", c2.dev, c2.ino);
        fts_assert (0);
    }
    close (cwd_fd);
    cwd_fd = parent_fd;
}
}
close (cwd_fd);
}
#endif

static unsigned short int
internal_function
fts_stat(FTS *sp, register FTSENT *p, bool follow)
{
    struct stat *sbp = p->fts_statp;

    if (ISSET (FTS_LOGICAL)
        || (ISSET (FTS_COMFOLLOW) && p->fts_level == FTS_ROOTLEVEL))
        follow = true;

    /*
     * If doing a logical walk, or application requested FTS_FOLLOW, do
     * a stat(2).  If that fails, check for a nonexistent symlink.  If
     * fail, set the errno from the stat call.
     */
    int flags = follow ? 0 : AT_SYMLINK_NOFOLLOW;
    if (fstatat (sp->fts_cwd_fd, p->fts_accpath, sbp, flags) < 0)
    {
        if (follow && errno == ENOENT
            && 0 <= fstatat (sp->fts_cwd_fd, p->fts_accpath, sbp,
                             AT_SYMLINK_NOFOLLOW))
        {
            __set_errno (0);
            return FTS_SLNONE;
        }

        p->fts_errno = errno;
        memset (sbp, 0, sizeof *sbp);
        return FTS_NS;
    }

    if (S_ISDIR(sbp->st_mode)) {
        if (!ISDOT(p->fts_name)) {
            /* Command-line "." and ".." are real directories. */
            return (p->fts_level == FTS_ROOTLEVEL ? FTS_D : FTS_DOT);
        }

        return (FTS_D);
    }
    if (S_ISLNK(sbp->st_mode))
        return (FTS_SL);
    if (S_ISREG(sbp->st_mode))
        return (FTS_F);
    return (FTS_DEFAULT);
}

```

```

static int
fts_compar (void const *a, void const *b)
{
/* Convert A and B to the correct types, to pacify the compiler, and
   for portability to bizarre hosts where "void const *" and "FTSENT
   const **" differ in runtime representation. The comparison
   function cannot modify *a and *b, but there is no compile-time
   check for this. */
FTSENT const **pa = (FTSENT const **) a;
FTSENT const **pb = (FTSENT const **) b;
return pa[0]->fts_fts->fts_compar (pa, pb);
}

static FTSENT *
internal_function
fts_sort (FTS *sp, FTSENT *head, register size_t nitems)
{
    register FTSENT **ap, *p;

    /* On most modern hosts, void * and FTSENT ** have the same
       run-time representation, and one can convert sp->fts_compar to
       the type qsort expects without problem. Use the heuristic that
       this is OK if the two pointer types are the same size, and if
       converting FTSENT ** to long int is the same as converting
       FTSENT ** to void * and then to long int. This heuristic isn't
       valid in general but we don't know of any counterexamples. */
    FTSENT *dummy;
    int (*compare) (void const *, void const *) =
    ((sizeof &dummy == sizeof (void *))
     && (long int) &dummy == (long int) (void *) &dummy)
     ? (int (*) (void const *, void const *)) sp->fts_compar
     : fts_compar;

    /*
     * Construct an array of pointers to the structures and call qsort(3).
     * Reassemble the array in the order returned by qsort. If unable to
     * sort for memory reasons, return the directory entries in their
     * current order. Allocate enough space for the current needs plus
     * 40 so don't realloc one entry at a time.
     */
    if (nitems > sp->fts_nitems) {
        FTSENT **a;

        sp->fts_nitems = nitems + 40;
        if (SIZE_MAX / sizeof *a < sp->fts_nitems
            || ! (a = realloc (sp->fts_array,
                               sp->fts_nitems * sizeof *a))) {
            free(sp->fts_array);
            sp->fts_array = NULL;
            sp->fts_nitems = 0;
            return (head);
        }
        sp->fts_array = a;
    }
    for (ap = sp->fts_array, p = head; p; p = p->fts_link)
        *ap++ = p;
    qsort((void *)sp->fts_array, nitems, sizeof(FTSENT *), compare);
    for (head = *(ap = sp->fts_array); --nitems; ++ap)
        ap[0]->fts_link = ap[1];
    ap[0]->fts_link = NULL;
}

```

```

        return (head);
    }

static FTSENT *
internal_function
fts_alloc (FTS *sp, const char *name, register size_t namelen)
{
    register FTSENT *p;
    size_t len;

/*
 * The file name is a variable length array. Allocate the FTSENT
 * structure and the file name in one chunk.
 */
len = FLEXSIZEOF(FTSENT, fts_name, namelen + 1);
if ((p = malloc(len)) == NULL)
    return (NULL);

/* Copy the name and guarantee NUL termination.*/
memcpy(p->fts_name, name, namelen);
p->fts_name[namelen] = '\0';

p->fts_namelen = namelen;
p->fts_fts = sp;
p->fts_path = sp->fts_path;
p->fts_errno = 0;
p->fts_dirp = NULL;
p->fts_flags = 0;
p->fts_instr = FTS_NOINSTR;
p->fts_number = 0;
p->fts_pointer = NULL;
return (p);
}

static void
internal_function
fts_lfree (register FTSENT *head)
{
    register FTSENT *p;
    int err = errno;

/* Free a linked list of structures.*/
while ((p = head)) {
    head = head->fts_link;
    if (p->fts_dirp)
        closedir (p->fts_dirp);
    free(p);
}

__set_errno (err);
}

/*
 * Allow essentially unlimited file name lengths; find, rm, ls should
 * all work on any tree. Most systems will allow creation of file
 * names much longer than MAXPATHLEN, even though the kernel won't
 * resolve them. Add the size (not just what's needed) plus 256 bytes
 * so don't realloc the file name 2 bytes at a time.
*/
static bool

```

```

internal_function
fts_palloc (FTS *sp, size_t more)
{
    char *p;
    size_t new_len = sp->fts_pathlen + more + 256;

    /*
     * See if fts_pathlen would overflow.
     */
    if (new_len < sp->fts_pathlen) {
        free(sp->fts_path);
        sp->fts_path = NULL;
        __set_errno (ENAMETOOLONG);
        return false;
    }
    sp->fts_pathlen = new_len;
    p = realloc(sp->fts_path, sp->fts_pathlen);
    if (p == NULL) {
        free(sp->fts_path);
        sp->fts_path = NULL;
        return false;
    }
    sp->fts_path = p;
    return true;
}

/*
 * When the file name is realloc'd, have to fix all of the pointers in
 * structures already returned.
 */
static void
internal_function
fts_padjust (FTS *sp, FTSENT *head)
{
    FTSENT *p;
    char *addr = sp->fts_path;

    /* This code looks at bit-patterns of freed pointers to
     * relocate them, so it relies on undefined behavior. If this
     * trick does not work on your platform, please report a bug. */

#define ADJUST(p) do { \
    uintptr_t old_accpath = (uintptr_t) (p)->fts_accpath; \
    if (old_accpath != (uintptr_t) (p)->fts_name) { \
        (p)->fts_accpath = \
            addr + (old_accpath - (uintptr_t) (p)->fts_path); \
    } \
    (p)->fts_path = addr; \
} while (0)
/* Adjust the current set of children. */
for (p = sp->fts_child; p; p = p->fts_link)
    ADJUST(p);

/* Adjust the rest of the tree, including the current level. */
for (p = head; p->fts_level >= FTS_ROOTLEVEL;) {
    ADJUST(p);
    p = p->fts_link ? p->fts_link : p->fts_parent;
}
}

```

```

static size_t
internal_function _GL_ATTRIBUTE_PURE
fts_maxarglen (char * const *argv)
{
    size_t len, max;

    for (max = 0; *argv; ++argv)
        if ((len = strlen(*argv)) > max)
            max = len;
    return (max + 1);
}

/*
 * Change to dir specified by fd or file name without getting
 * tricked by someone changing the world out from underneath us.
 * Assumes p->fts_statp->st_dev and p->fts_statp->st_ino are filled in.
 * If FD is non-negative, expect it to be used after this function returns,
 * and to be closed eventually. So don't pass e.g., 'dirfd(dirp)' and then
 * do closedir(dirp), because that would invalidate the saved FD.
 * Upon failure, close FD immediately and return nonzero.
*/
static int
internal_function
fts_safe_changedir (FTS *sp, FTSENT *p, int fd, char const *dir)
{
    int ret;
    bool is_dotdot = dir && STREQ (dir, "..");
    int newfd;

    /* This clause handles the unusual case in which FTS_NOCHDIR
     * is specified, along with FTS_CWD. In that case, there is
     * no need to change even the virtual cwd file descriptor.
     * However, if FD is non-negative, we do close it here. */
    if (ISSET (FTS_NOCHDIR))
    {
        if (ISSET (FTS_CWD) && 0 <= fd)
            close (fd);
        return 0;
    }

    if (fd < 0 && is_dotdot && ISSET (FTS_CWD))
    {
        /* When possible, skip the diopen and subsequent fstat+dev/ino
         * comparison. I.e., when changing to parent directory
         * (chdir ("..")), use a file descriptor from the ring and
         * save the overhead of diopen+fstat, as well as avoiding
         * failure when we lack "x" access to the virtual cwd. */
        if (! i_ring_empty (&sp->fts_fd_ring))
        {
            int parent_fd;
            fd_ring_print (sp, stderr, "pre-pop");
            parent_fd = i_ring_pop (&sp->fts_fd_ring);
            if (0 <= parent_fd)
            {
                fd = parent_fd;
                dir = NULL;
            }
        }
    }
}

```

```

newfd = fd;
if (fd < 0 && (newfd = diropen (sp, dir)) < 0)
return -1;

/* The following dev/inode check is necessary if we're doing a
"logical" traversal (through symlinks, a la chown -L), if the
system lacks O_NOFOLLOW support, or if we're changing to ".."
(but not via a popped file descriptor). When changing to the
name "..", O_NOFOLLOW can't help. In general, when the target is
not "..", diropen's use of O_NOFOLLOW ensures we don't mistakenly
follow a symlink, so we can avoid the expense of this fstat. */
if (ISSET(FTS_LOGICAL) || ! HAVE_WORKING_O_NOFOLLOW
    || (dir && STREQ (dir, "..")))
{
    struct stat sb;
    if (fstat(newfd, &sb))
    {
        ret = -1;
        goto bail;
    }
    if (p->fts_statp->st_dev != sb.st_dev
        || p->fts_statp->st_ino != sb.st_ino)
    {
        __set_errno (ENOENT);      /* disinformation */
        ret = -1;
        goto bail;
    }
}

if (ISSET(FTS_CWDFFD))
{
    cwd_advance_fd (sp, newfd, ! is_dotdot);
    return 0;
}

ret = fchdir(newfd);
bail:
if (fd < 0)
{
    int oerrno = errno;
    (void)close(newfd);
    __set_errno (oerrno);
}
return ret;
"""

,
"patche":
"""

diff --git a/lib/fts.c b/lib/fts.c
index 5a86419..d99e666 100644
--- a/lib/fts.c
+++ b/lib/fts.c
@@ -1394,8 +1394,11 @@ fts_build (register FTS *sp, int type)
        != NO_LEAF_OPTIMIZATION));
    if (descend || type == BREAD)
    {
-       if (ISSET(FTS_CWDFFD))
+       if (ISSET(FTS_CWDFFD)) {
+           int old_fd = dir_fd;
         dir_fd = fcntl (dir_fd, F_DUPFD_CLOEXEC, STDERR_FILENO + 1);
    }
}

```

```

+           close(old_fd); // avoid lingering open file descriptors
+
+       }
+       if (dir_fd < 0 || fts_safe_changedir(sp, cur, dir_fd, NULL)) {
+           if (descend && type == BREAD)
+               cur->fts_errno = errno;
+       }
+   },
+
{
    "wrong_code": [
        /* Program name management.
Copyright (C) 2016-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation; either version 2.1 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */

```

```

#include <config.h>

/* Specification. Also get __argv declaration. */
#include <stdlib.h>

```

```
#include <errno.h> /* get program_invocation_name declaration */
```

```
#ifdef __AIX
# include <unistd.h>
# include <procinfo.h>
# include <string.h>
#endif
```

```
#ifdef __MVS__
# ifndef __OPEN_SYS
#  define __OPEN_SYS
# endif
# include <string.h>
# include <sys/ps.h>
#endif
```

```
#ifdef __hpux
# include <unistd.h>
# include <sys/param.h>
# include <sys/pstat.h>
# include <string.h>
#endif
```

```
#if defined __sgi || defined __osf__
# include <string.h>
# include <unistd.h>
# include <stdio.h>
# include <fcntl.h>
```

```

# include <sys/procfs.h>
#endif

#if defined __SCO_VERSION__ || defined __sysv5__
# include <fcntl.h>
# include <string.h>
#endif

#include "basename-igpl.h"

#ifndef HAVE_GETPROGNAME /* not Mac OS X, FreeBSD, NetBSD, OpenBSD >= 5.4,
Solaris >= 11, Cygwin, Android API level >= 21 */
char const *
getprogname (void)
{
# if HAVE_DECL_PROGRAM_INVOCATION_SHORT_NAME           /* glibc, BeOS */
/* https://www.gnu.org/software/libc/manual/html_node/Error-Messages.html */
return program_invocation_short_name;
# elif HAVE_DECL_PROGRAM_INVOCATION_NAME               /* glibc, BeOS */
/* https://www.gnu.org/software/libc/manual/html_node/Error-Messages.html */
return last_component (program_invocation_name);
# elif HAVE_GETEXECNAME                                /* Solaris */
/* https://docs.oracle.com/cd/E19253-01/816-5168/6mbb3hrb1/index.html */
const char *p = getexecname ();
if (!p)
    p = "?";
return last_component (p);
# elif HAVE_DECL__ARGV                                /* mingw, MSVC */
/* https://docs.microsoft.com/en-us/cpp/c-runtime-library/argc-argv-wargv */
const char *p = __argv && __argv[0] ? __argv[0] : "?";
return last_component (p);
# elif HAVE_VAR__PROGNAME                            /* OpenBSD, Android, QNX */
/* https://man.openbsd.org/style.9 */
/* http://www.qnx.de/developers/docs/6.5.0/index.jsp?
topic=%2Fcom.qnx.doc.neutrino_lib_ref%2Fp%2F__progname.html */
/* Be careful to declare this only when we absolutely need it
(OpenBSD 5.1), rather than when it's available. Otherwise,
its mere declaration makes program_invocation_short_name
malfunction (have zero length) with Fedora 25's glibc. */
extern char *__progname;
const char *p = __progname;
# if defined __ANDROID__
return last_component (p);
# else
return p && p[0] ? p : "?";
# endif
# elif __AIX                                         /* AIX */
/* Idea by Bastien ROUCARIÈS,
https://lists.gnu.org/r/bug-gnulib/2010-12/msg00095.html
Reference: https://www.ibm.com/support/knowledgecenter/en/ssw_aix_61/
com.ibm.aix.basetrf1/getprocs.htm
*/
static char *p;
static int first = 1;
if (first)
{
    first = 0;
    pid_t pid = getpid ();
    struct procentry64 procs;
    p = (0 < getprocs64 (&procs, sizeof procs, NULL, 0, &pid, 1)

```

```

    ? strdup (procs.pi_comm)
    : NULL);
if (!p)
    p = "?";
}
return p;
#endif defined __hpux
static char *p;
static int first = 1;
if (first)
{
    first = 0;
    pid_t pid = getpid ();
    struct pst_status status;
    if (pstat_getproc (&status, sizeof status, 0, pid) > 0)
    {
        char *ucomm = status.pst_ucomm;
        char *cmd = status.pst_cmd;
        if (strlen (ucomm) < PST_UCOMMLEN - 1)
            p = ucomm;
        else
        {
            /* ucomm is truncated to length PST_UCOMMLEN - 1.
               Look at cmd instead. */
            char *space = strchr (cmd, ' ');
            if (space != NULL)
                *space = '\0';
            p = strrchr (cmd, '/');
            if (p != NULL)
                p++;
            else
                p = cmd;
            if (strlen (p) > PST_UCOMMLEN - 1
                && memcmp (p, ucomm, PST_UCOMMLEN - 1) == 0)
                /* p is less truncated than ucomm. */
                ;
            else
                p = ucomm;
        }
        p = strdup (p);
    }
}
else
{
#endif if !defined __LP64__
/* Support for 32-bit programs running in 64-bit HP-UX.
   The documented way to do this is to use the same source code
   as above, but in a compilation unit where '#define _PSTAT64 1'
   is in effect. I prefer a single compilation unit; the struct
   size and the offsets are not going to change. */
char status64[1216];
if (__pstat_getproc64 (status64, sizeof status64, 0, pid) > 0)
{
    char *ucomm = status64 + 288;
    char *cmd = status64 + 168;
    if (strlen (ucomm) < PST_UCOMMLEN - 1)
        p = ucomm;
    else
    {
        /* ucomm is truncated to length PST_UCOMMLEN - 1.
           Look at cmd instead. */

```

```

char *space = strchr (cmd, ' ');
if (space != NULL)
    *space = '\0';
p = strrchr (cmd, '/');
if (p != NULL)
    p++;
else
    p = cmd;
if (strlen (p) > PST_UCOMMLEN - 1
    && memcmp (p, ucomm, PST_UCOMMLEN - 1) == 0)
    /* p is less truncated than ucomm. */
    ;
else
    p = ucomm;
}
p = strdup (p);
}
else
{
    p = NULL;
}
if (!p)
    p = "?";
}
return p;
# elif __MVS__                                /* z/OS */
/* https://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/
com.ibm.zos.v2r1.bpxbd00/rtwgetp.htm */
static char *p = "?";
static int first = 1;
if (first)
{
    pid_t pid = getpid ();
    int token;
    W_PSPROC buf;
    first = 0;
    memset (&buf, 0, sizeof(buf));
    buf.ps_cmddptr = (char *) malloc (buf.ps_cmddlen = PS_CMDBLEN_LONG);
    buf.ps_conttyptr = (char *) malloc (buf.ps_conttylen = PS_CONTTYBLEN);
    buf.ps_pathptr = (char *) malloc (buf.ps_pathlen = PS_PATHBLEN);
    if (buf.ps_cmddptr && buf.ps_conttyptr && buf.ps_pathptr)
    {
        for (token = 0; token >= 0;
            token = w_getpsent (token, &buf, sizeof(buf)))
        {
            if (token > 0 && buf.ps_pid == pid)
            {
                char *s = strdup (last_component (buf.ps_pathptr));
                if (s)
                {
# if defined __XPLINK__ && __CHARSET_LIB == 1
                    /* The compiler option -qascii is in use.
                     https://makingdeveloperslivesbetter.wordpress.com/2022/01/07/is-z-os-
ascii-or-ebcdic-yes/
                     https://www.ibm.com/docs/en/zos/2.5.0?topic=features-macros-related-
compiler-option-settings
                     So, convert the result from EBCDIC to ASCII.
                     https://www.ibm.com/docs/en/zos/2.5.0?topic=functions-e2a-s-convert-
string-from-ebcdic-ascii */
                    if (__e2a_s (s) == (size_t)-1)

```

```

        free (s);
    else
# endif
        p = s;
    }
break;
}
}
}
free (buf.ps_cmdptr);
free (buf.ps_conttyptr);
free (buf.ps_pathptr);
}
return p;
# elif defined __sgi || defined __osf__           /* IRIX or Tru64 */
char filename[50];
int fd;

# if defined __sgi
    sprintf (filename, "/proc/pinfo/%d", (int) getpid ());
# else
    sprintf (filename, "/proc/%d", (int) getpid ());
# endif
fd = open (filename, O_RDONLY | O_CLOEXEC);
if (0 <= fd)
{
    prpsinfo_t buf;
    int ioctl_ok = 0 <= ioctl (fd, PIOCPSPINFO, &buf);
    close (fd);
    if (ioctl_ok)
    {
        char *name = buf.pr_fname;
        size_t namesize = sizeof buf.pr_fname;
        /* It may not be NUL-terminated. */
        char *namenul = memchr (name, '\0', namesize);
        size_t namelen = namenul ? namenul - name : namesize;
        char *namecopy = malloc (namelen + 1);
        if (namecopy)
        {
            namecopy[namelen] = '\0';
            return memcpy (namecopy, name, namelen);
        }
    }
}
return NULL;
# elif defined __SCO_VERSION__ || defined __sysv5__      /* SCO OpenServer6/
UnixWare */
char buf[80];
int fd;
sprintf (buf, "/proc/%d/cmdline", getpid());
fd = open (buf, O_RDONLY);
if (0 <= fd)
{
    size_t n = read (fd, buf, 79);
    if (n > 0)
    {
        buf[n] = '\0'; /* Guarantee null-termination */
        char *progname;
        progname = strrchr (buf, '/');
        if (progname)

```

```

    {
    progname = progname + 1; /* Skip the '/' */
    }
else
{
    progname = buf;
}
char *ret;
ret = malloc (strlen (progname) + 1);
if (ret)
{
    strcpy (ret, progname);
    return ret;
}
close (fd);
}
return "?";
#else
#error "getprogname module not ported to this OS"
#endif
}

#endif

/*
 * Hey Emacs!
 * Local Variables:
 * coding: utf-8
 * End:
 */
"""

,
"error_category": "Encoding Mismatch",
"error": "Program name contains invalid characters or appears garbled.",
"correct_code": """
"""

/* Program name management.
Copyright (C) 2016-2024 Free Software Foundation, Inc.
```

This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU Lesser General Public License as published by  
the Free Software Foundation; either version 2.1 of the License, or  
(at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License  
along with this program. If not, see <<https://www.gnu.org/licenses/>>. \*/

```
#include <config.h>

/* Specification. Also get __argv declaration. */
#include <stdlib.h>

#include <errno.h> /* get program_invocation_name declaration */

#ifndef _AIX
```

```

# include <unistd.h>
# include <procinfo.h>
# include <string.h>
#endif

#ifndef __MVS__
# ifndef _OPEN_SYS
# define _OPEN_SYS
# endif
# include <string.h>
# include <sys/ps.h>
#endif

#ifndef __hpux
# include <unistd.h>
# include <sys/param.h>
# include <sys/pstat.h>
# include <string.h>
#endif

#if defined __sgi || defined __osf__
# include <string.h>
# include <unistd.h>
# include <stdio.h>
# include <fcntl.h>
# include <sys/procfs.h>
#endif

#if defined __SCO_VERSION__ || defined __sysv5__
# include <fcntl.h>
# include <string.h>
#endif

#include "basename-lgpl.h"

#ifndef HAVE_GETPROGNAME /* not Mac OS X, FreeBSD, NetBSD, OpenBSD >= 5.4,
Solaris >= 11, Cygwin, Android API level >= 21 */
char const *
getprogname (void)
{
# if HAVE_DECL_PROGRAM_INVOCATION_SHORT_NAME /* glibc, BeOS */
/* https://www.gnu.org/software/libc/manual/html_node/Error-Messages.html */
return program_invocation_short_name;
# elif HAVE_DECL_PROGRAM_INVOCATION_NAME /* glibc, BeOS */
/* https://www.gnu.org/software/libc/manual/html_node/Error-Messages.html */
return last_component (program_invocation_name);
# elif HAVE_GETEXECNAME /* Solaris */
/* https://docs.oracle.com/cd/E19253-01/816-5168/6mbb3hrb1/index.html */
const char *p = getexecname ();
if (!p)
  p = "?";
return last_component (p);
# elif HAVE_DECL__ARGV /* mingw, MSVC */
/* https://docs.microsoft.com/en-us/cpp/c-runtime-library/argc-argv-wargv */
const char *p = __argv && __argv[0] ? __argv[0] : "?";
return last_component (p);
# elif HAVE_VAR__PROGNAME /* OpenBSD, Android, QNX */
/* https://man.openbsd.org/style.9 */
/* http://www.qnx.de/developers/docs/6.5.0/index.jsp?
topic=%2Fcom.qnx.doc.neutrino_lib_ref%2Fp%2F__progname.html */

```

```

/* Be careful to declare this only when we absolutely need it
   (OpenBSD 5.1), rather than when it's available. Otherwise,
   its mere declaration makes program_invocation_short_name
   malfunction (have zero length) with Fedora 25's glibc. */
extern char *__progname;
const char *p = __progname;
# if defined __ANDROID__
return last_component (p);
# else
return p && p[0] ? p : "?";
# endif
# elif __AIX                               /* AIX */
/* Idea by Bastien ROUCARIÈS,
   https://lists.gnu.org/r/bug-gnulib/2010-12/msg00095.html
   Reference: https://www.ibm.com/support/knowledgecenter/en/ssw_aix_61/
com.ibm.aix.basetrf1/getprocs.htm
*/
static char *p;
static int first = 1;
if (first)
{
    first = 0;
    pid_t pid = getpid ();
    struct procentry64 procs;
    p = (0 < getprocs64 (&procs, sizeof procs, NULL, 0, &pid, 1)
        ? strdup (procs.pi_comm)
        : NULL);
    if (!p)
        p = "?";
}
return p;
# elif defined __hpux
static char *p;
static int first = 1;
if (first)
{
    first = 0;
    pid_t pid = getpid ();
    struct pst_status status;
    if (pstat_getproc (&status, sizeof status, 0, pid) > 0)
    {
        char *ucomm = status.pst_ucomm;
        char *cmd = status.pst_cmd;
        if (strlen (ucomm) < PST_UCOMMLEN - 1)
            p = ucomm;
        else
        {
            /* ucomm is truncated to length PST_UCOMMLEN - 1.
               Look at cmd instead. */
            char *space = strchr (cmd, ' ');
            if (space != NULL)
                *space = '\0';
            p = strrchr (cmd, '/');
            if (p != NULL)
                p++;
            else
                p = cmd;
            if (strlen (p) > PST_UCOMMLEN - 1
                && memcmp (p, ucomm, PST_UCOMMLEN - 1) == 0)
                /* p is less truncated than ucomm. */

```

```

        ;
    else
        p = ucomm;
    }
    p = strdup (p);
}
else
{
# if !defined __LP64__
/* Support for 32-bit programs running in 64-bit HP-UX.
The documented way to do this is to use the same source code
as above, but in a compilation unit where '#define _PSTAT64 1'
is in effect. I prefer a single compilation unit; the struct
size and the offsets are not going to change. */
char status64[1216];
if (__pstat_getproc64 (status64, sizeof status64, 0, pid) > 0)
{
    char *ucomm = status64 + 288;
    char *cmd = status64 + 168;
    if (strlen (ucomm) < PST_UCOMMLEN - 1)
        p = ucomm;
    else
    {
        /* ucomm is truncated to length PST_UCOMMLEN - 1.
        Look at cmd instead. */
        char *space = strchr (cmd, ' ');
        if (space != NULL)
            *space = '\0';
        p = strrchr (cmd, '/');
        if (p != NULL)
            p++;
        else
            p = cmd;
        if (strlen (p) > PST_UCOMMLEN - 1
            && memcmp (p, ucomm, PST_UCOMMLEN - 1) == 0)
            /* p is less truncated than ucomm. */
            ;
        else
            p = ucomm;
    }
    p = strdup (p);
}
else
# endif
    p = NULL;
}
if (!p)
    p = "?";
}
return p;
# elif __MVS__                                /* z/OS */
/* https://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/
com.ibm.zos.v2r1.bpxbd00/rtwgetp.htm */
static char *p = "?";
static int first = 1;
if (first)
{
    pid_t pid = getpid ();
    int token;
    W_PSPROC buf;

```

```

first = 0;
memset (&buf, 0, sizeof(buf));
buf.ps_cmdbuf = (char *) malloc (buf.ps_cmrlen = PS_CMDBLEN_LONG);
buf.ps_contytptr = (char *) malloc (buf.ps_conttylen = PS_CONTTYBLEN);
buf.ps_pathptr = (char *) malloc (buf.ps_pathlen = PS_PATHBLEN);
if (buf.ps_cmdbuf && buf.ps_contytptr && buf.ps_pathptr)
{
    for (token = 0; token >= 0;
        token = w_getpsent (token, &buf, sizeof(buf)))
    {
        if (token > 0 && buf.ps_pid == pid)
        {
            char *s = strdup (last_component (buf.ps_pathptr));
            if (s)
            {
                p = s;
            }
            break;
        }
    }
    free (buf.ps_cmdbuf);
    free (buf.ps_contytptr);
    free (buf.ps_pathptr);
}
return p;
# elif defined __sgi || defined __osf__                                /* IRIX or Tru64 */
char filename[50];
int fd;

# if defined __sgi
    sprintf (filename, "/proc/pinfo/%d", (int) getpid ());
# else
    sprintf (filename, "/proc/%d", (int) getpid ());
# endif
fd = open (filename, O_RDONLY | O_CLOEXEC);
if (0 <= fd)
{
    prpsinfo_t buf;
    int ioctl_ok = 0 <= ioctl (fd, PIOCPSPINFO, &buf);
    close (fd);
    if (ioctl_ok)
    {
        char *name = buf.pr_fname;
        size_t namesize = sizeof buf.pr_fname;
        /* It may not be NUL-terminated. */
        char *namenul = memchr (name, '\0', namesize);
        size_t namelen = namenul ? namenul - name : namesize;
        char *namecopy = malloc (namelen + 1);
        if (namecopy)
        {
            namecopy[namelen] = '\0';
            return memcpy (namecopy, name, namelen);
        }
    }
}
return NULL;
# elif defined __SCO_VERSION__ || defined __sysv5__                  /* SCO OpenServer6/
UnixWare */
char buf[80];

```

```

int fd;
sprintf (buf, "/proc/%d/cmdline", getpid());
fd = open (buf, O_RDONLY);
if (0 <= fd)
{
    size_t n = read (fd, buf, 79);
    if (n > 0)
    {
        buf[n] = '\0'; /* Guarantee null-termination */
        char *progname;
        progname = strrchr (buf, '/');
        if (progname)
        {
            progname = progname + 1; /* Skip the '/' */
        }
        else
        {
            progname = buf;
        }
    }
    char *ret;
    ret = malloc (strlen (progname) + 1);
    if (ret)
    {
        strcpy (ret, progname);
        return ret;
    }
}
close (fd);
}
return "?";
#else
#error "getprogname module not ported to this OS"
#endif
}

#endif

/*
 * Hey Emacs!
 * Local Variables:
 * coding: utf-8
 * End:
 */
"""
"patch":
"""

diff --git a/lib/getprogname.c b/lib/getprogname.c
index 204855a..5eada0b 100644
--- a/lib/getprogname.c
+++ b/lib/getprogname.c
@@ -213,16 +213,6 @@ getprogname (void)
    char *s = strdup (last_component (buf.ps_pathptr));
    if (s)
    {
-# if defined __XPLINK__ && __CHARSET_LIB == 1
-/* The compiler option -qascii is in use.
-   https://makingdeveloperslivesbetter.wordpress.com/2022/01/07/is-z-os-
ascii-or-ebcdic-yes/
-   https://www.ibm.com/docs/en/zos/2.5.0?topic=features-macros-related-
compiler-option-settings

```

```

        - So, convert the result from EBCDIC to ASCII.
        - https://www.ibm.com/docs/en/zos/2.5.0?topic=functions-e2a-s-convert-
string-from-ebcdic-ascii */
        - if (__e2a_s (s) == (size_t)-1)
        -     free (s);
        - else
-# endif
        p = s;
    }
break;
"""
},
{
  "wrong_code": """
/* Detect the number of processors.
Copyright (C) 2009-2024 Free Software Foundation, Inc.
This file is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation; either version 2.1 of the
License, or (at your option) any later version.

This file is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */
/* Written by Glen Lenker and Bruno Haible. */

#include <config.h>
#include "nproc.h"

#include <limits.h>
#include <stdlib.h>
#include <unistd.h>

#if HAVE_PTHREAD_GETAFFINITY_NP && 0
# include <pthread.h>
# include <sched.h>
#endif
#if HAVE_SCHED_GETAFFINITY_LIKE_GLIBC || HAVE_SCHED_GETAFFINITY_NP
# include <sched.h>
#endif

#include <sys/types.h>

#if HAVE_SYS_PSTAT_H
# include <sys/pstat.h>
#endif

#if HAVE_SYS_SYSMP_H
# include <sys/sysmp.h>
#endif

#if HAVE_SYS_PARAM_H
# include <sys/param.h>

```

```

#endif

#if HAVE_SYS_SYSCTL_H && !(defined __GLIBC__ && defined __linux__)
# include <sys/sysctl.h>
#endif

#if defined _WIN32 && ! defined __CYGWIN__
# define WIN32_LEAN_AND_MEAN
# include <windows.h>
#endif

#include "c-ctype.h"

#include "minmax.h"

#define ARRAY_SIZE(a) (sizeof (a) / sizeof ((a)[0]))

/* Return the number of processors available to the current process, based
on a modern system call that returns the "affinity" between the current
process and each CPU. Return 0 if unknown or if such a system call does
not exist. */
static unsigned long
num_processors_via_affinity_mask (void)
{
/* glibc >= 2.3.3 with NPTL and NetBSD 5 have pthread_getaffinity_np,
but with different APIs. Also it requires linking with -lpthread.
Therefore this code is not enabled.
glibc >= 2.3.4 has sched_getaffinity whereas NetBSD 5 has
sched_getaffinity_np. */
#ifndef HAVE_PTHREAD_GETAFFINITY_NP && defined __GLIBC__ && 0
{
    cpu_set_t set;

    if (pthread_getaffinity_np (pthread_self (), sizeof (set), &set) == 0)
    {
        unsigned long count;

        # ifdef CPU_COUNT
        /* glibc >= 2.6 has the CPU_COUNT macro. */
        count = CPU_COUNT (&set);
        # else
        size_t i;

        count = 0;
        for (i = 0; i < CPU_SETSIZE; i++)
            if (CPU_ISSET (i, &set))
                count++;
        # endif
        if (count > 0)
            return count;
    }
}
#endif HAVE_PTHREAD_GETAFFINITY_NP && defined __NetBSD__ && 0
{
    cpuset_t *set;

    set = cpuset_create ();
    if (set != NULL)
    {
        unsigned long count = 0;

```



```

        break;
        if (ret > 0)
            count++;
    }
}
cpuset_destroy (set);
if (count > 0)
    return count;
}
#endif

#if defined _WIN32 && ! defined __CYGWIN__
{ /* This works on native Windows platforms. */
    DWORD_PTR process_mask;
    DWORD_PTR system_mask;

    if (GetProcessAffinityMask (GetCurrentProcess (),
                               &process_mask, &system_mask))
    {
        DWORD_PTR mask = process_mask;
        unsigned long count = 0;

        for (; mask != 0; mask = mask >> 1)
            if (mask & 1)
                count++;
        if (count > 0)
            return count;
    }
}
#endif

return 0;
}

```

/\* Return the total number of processors. Here QUERY must be one of NPROC\_ALL, NPROC\_CURRENT. The result is guaranteed to be at least 1. \*/

```

static unsigned long int
num_processors_ignoring_omp (enum nproc_query query)
{
    /* On systems with a modern affinity mask system call, we have
       sysconf (_SC_NPROCESSORS_CONF)
       >= sysconf (_SC_NPROCESSORS_ONLN)
       >= num_processors_via_affinity_mask ()
    The first number is the number of CPUs configured in the system.
    The second number is the number of CPUs available to the scheduler.
    The third number is the number of CPUs available to the current process.

```

Note! On Linux systems with glibc, the first and second number come from the /sys and /proc file systems (see glibc/sysdeps/unix/sysv/linux/getsysstats.c). In some situations these file systems are not mounted, and the sysconf call returns 1 or 2 (<[https://sourceware.org/bugzilla/show\\_bug.cgi?id=21542](https://sourceware.org/bugzilla/show_bug.cgi?id=21542)>), which does not reflect the reality. \*/

```

if (query == NPROC_CURRENT)
{
    /* Try the modern affinity mask system call. */
    {

```

```

        unsigned long nprocs = num_processors_via_affinity_mask ();

        if (nprocs > 0)
            return nprocs;
    }

#if defined _SC_NPROCESSORS_ONLN
{ /* This works on glibc, Mac OS X 10.5, FreeBSD, AIX, OSF/1, Solaris,
   Cygwin, Haiku. */
    long int nprocs = sysconf (_SC_NPROCESSORS_ONLN);
    if (nprocs > 0)
        return nprocs;
}
#endif
else /* query == NPROC_ALL */
{
#if defined _SC_NPROCESSORS_CONF
{ /* This works on glibc, Mac OS X 10.5, FreeBSD, AIX, OSF/1, Solaris,
   Cygwin, Haiku. */
    long int nprocs = sysconf (_SC_NPROCESSORS_CONF);

# if __GLIBC__ >= 2 && defined __linux__
    /* On Linux systems with glibc, this information comes from the /sys and
     /proc file systems (see glibc/sysdeps/unix/linux/getsysstats.c).
     In some situations these file systems are not mounted, and the
     sysconf call returns 1 or 2. But we wish to guarantee that
     num_processors (NPROC_ALL) >= num_processors (NPROC_CURRENT). */
    if (nprocs == 1 || nprocs == 2)
    {
        unsigned long nprocs_current = num_processors_via_affinity_mask ();

        if /* nprocs_current > 0 && */ nprocs_current > nprocs)
            nprocs = nprocs_current;
    }
# endif
    if (nprocs > 0)
        return nprocs;
}
#endif
}

#endif HAVE_PSTAT_GETDYNAMIC
{ /* This works on HP-UX. */
    struct pst_dynamic psd;
    if (pststat_getdynamic (&psd, sizeof psd, 1, 0) >= 0)
    {
        /* The field psd_proc_cnt contains the number of active processors.
         In newer releases of HP-UX 11, the field psd_max_proc_cnt includes
         deactivated processors. */
        if (query == NPROC_CURRENT)
        {
            if (psd.psd_proc_cnt > 0)
                return psd.psd_proc_cnt;
        }
        else
        {
            if (psd.psd_max_proc_cnt > 0)
                return psd.psd_max_proc_cnt;
        }
    }
}
```

```

        }
    }
#endif

#if HAVE_SYSMP && defined MP_NAPROCS && defined MP_NPROCS
{ /* This works on IRIX. */
    /* MP_NPROCS yields the number of installed processors.
       MP_NAPROCS yields the number of processors available to unprivileged
       processes. */
    int nprocs =
        sysmp (query == NPROC_CURRENT && getuid () != 0
            ? MP_NAPROCS
            : MP_NPROCS);
    if (nprocs > 0)
        return nprocs;
}
#endif

/* Finally, as fallback, use the APIs that don't distinguish between
   NPROC_CURRENT and NPROC_ALL. */

#if HAVE_SYSCTL && !(defined __GLIBC__ && defined __linux__)
{ /* This works on macOS, FreeBSD, NetBSD, OpenBSD.
   macOS 10.14 does not allow mib to be const. */
    int nprocs;
    size_t len = sizeof (nprocs);
    static int mib[] [2] = {
        # ifdef HW_NCPUONLINE
            { CTL_HW, HW_NCPUONLINE },
        # endif
        { CTL_HW, HW_NCPU }
    };
    for (int i = 0; i < ARRAY_SIZE (mib); i++)
    {
        if (sysctl (mib[i], ARRAY_SIZE (mib[i]), &nprocs, &len, NULL, 0) == 0
            && len == sizeof (nprocs)
            && 0 < nprocs)
            return nprocs;
    }
}
#endif

#if defined _WIN32 && ! defined __CYGWIN__
{ /* This works on native Windows platforms. */
    SYSTEM_INFO system_info;
    GetSystemInfo (&system_info);
    if (0 < system_info.dwNumberOfProcessors)
        return system_info.dwNumberOfProcessors;
}
#endif

return 1;
}

/* Parse OMP environment variables without dependence on OMP.
   Return 0 for invalid values. */
static unsigned long int
parse_omp_threads (char const* threads)
{

```

```

unsigned long int ret = 0;

if (threads == NULL)
    return ret;

/* The OpenMP spec says that the value assigned to the environment variables
   "may have leading and trailing white space". */
while (*threads != '\0' && c_isspace (*threads))
    threads++;

/* Convert it from positive decimal to 'unsigned long'. */
if (c_isdigit (*threads))
{
    char *endptr = NULL;
    unsigned long int value = strtoul (threads, &endptr, 10);

    if (endptr != NULL)
    {
        while (*endptr != '\0' && c_isspace (*endptr))
            endptr++;
        if (*endptr == '\0')
            return value;
        /* Also accept the first value in a nesting level,
           since we can't determine the nesting level from env vars. */
        else if (*endptr == ',')
            return value;
    }
}

return ret;
}

unsigned long int
num_processors (enum nproc_query query)
{
unsigned long int omp_env_limit = ULONG_MAX;

if (query == NPROC_CURRENT_OVERRIDEABLE)
{
    unsigned long int omp_env_threads;
    /* Honor the OpenMP environment variables, recognized also by all
       programs that are based on OpenMP. */
    omp_env_threads = parse_omp_threads (getenv ("OMP_NUM_THREADS"));
    omp_env_limit = parse_omp_threads (getenv ("OMP_THREAD_LIMIT"));
    if (!omp_env_limit)
        omp_env_limit = ULONG_MAX;

    if (omp_env_threads)
        return MIN (omp_env_threads, omp_env_limit);

    query = NPROC_CURRENT;
}
/* Here query is one of NPROC_ALL, NPROC_CURRENT. */
{
    unsigned long nprocs = num_processors_ignoring_omp (query);
    return MIN (nprocs, omp_env_limit);
}
"""

,
"error_category": "Linker Errors",

```

```
"error": "Undefined reference to __get_num_online_cpus during linking.",  
"correct_code":  
"""  
/* Detect the number of processors.  
  
Copyright (C) 2009-2024 Free Software Foundation, Inc.  
  
This file is free software: you can redistribute it and/or modify  
it under the terms of the GNU Lesser General Public License as  
published by the Free Software Foundation; either version 2.1 of the  
License, or (at your option) any later version.  
  
This file is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU Lesser General Public License for more details.  
  
You should have received a copy of the GNU Lesser General Public License  
along with this program. If not, see <https://www.gnu.org/licenses/>. */  
  
/* Written by Glen Lenker and Bruno Haible. */  
  
#include <config.h>  
#include "nproc.h"  
  
#include <limits.h>  
#include <stdlib.h>  
#include <unistd.h>  
  
#if HAVE_PTHREAD_GETAFFINITY_NP && 0  
# include <pthread.h>  
# include <sched.h>  
#endif  
#if HAVE_SCHED_GETAFFINITY_LIKE_GLIBC || HAVE_SCHED_GETAFFINITY_NP  
# include <sched.h>  
#endif  
  
#include <sys/types.h>  
  
#if HAVE_SYS_PSTAT_H  
# include <sys/pstat.h>  
#endif  
  
#if HAVE_SYS_SYSMP_H  
# include <sys/sysmp.h>  
#endif  
  
#if HAVE_SYS_PARAM_H  
# include <sys/param.h>  
#endif  
  
#if HAVE_SYS_SYSCTL_H && !(defined __GLIBC__ && defined __linux__)  
# include <sys/sysctl.h>  
#endif  
  
#if defined _WIN32 && ! defined __CYGWIN__  
# define WIN32_LEAN_AND_MEAN  
# include <windows.h>  
#endif
```

```

#include "c-ctype.h"
#include "minmax.h"

#define ARRAY_SIZE(a) (sizeof (a) / sizeof ((a)[0]))

/* Return the number of processors available to the current process, based
on a modern system call that returns the "affinity" between the current
process and each CPU. Return 0 if unknown or if such a system call does
not exist. */
static unsigned long
num_processors_via_affinity_mask (void)
{
/* glibc >= 2.3.3 with NPTL and NetBSD 5 have pthread_getaffinity_np,
but with different APIs. Also it requires linking with -lpthread.
Therefore this code is not enabled.
glibc >= 2.3.4 has sched_getaffinity whereas NetBSD 5 has
sched_getaffinity_np. */
#if HAVE_PTHREAD_GETAFFINITY_NP && defined __GLIBC__ && 0
{
    cpu_set_t set;

    if (pthread_getaffinity_np (pthread_self (), sizeof (set), &set) == 0)
    {
        unsigned long count;

# ifdef CPU_COUNT
        /* glibc >= 2.6 has the CPU_COUNT macro. */
        count = CPU_COUNT (&set);
# else
        size_t i;

        count = 0;
        for (i = 0; i < CPU_SETSIZE; i++)
            if (CPU_ISSET (i, &set))
                count++;
# endif
        if (count > 0)
            return count;
    }
}
#elif HAVE_PTHREAD_GETAFFINITY_NP && defined __NetBSD__ && 0
{
    cpuset_t *set;

    set = cpuset_create ();
    if (set != NULL)
    {
        unsigned long count = 0;

        if (pthread_getaffinity_np (pthread_self (), cpuset_size (set), set)
            == 0)
        {
            cpuid_t i;

            for (i = 0;; i++)
            {
                int ret = cpuset_isset (i, set);
                if (ret < 0)
                    break;
            }
        }
    }
}

```



```

#ifndef _WIN32 && !defined __CYGWIN__
{ /* This works on native Windows platforms. */
    DWORD_PTR process_mask;
    DWORD_PTR system_mask;

    if (GetProcessAffinityMask (GetCurrentProcess (),
                               &process_mask, &system_mask))
    {
        DWORD_PTR mask = process_mask;
        unsigned long count = 0;

        for (; mask != 0; mask = mask >> 1)
            if (mask & 1)
                count++;
            if (count > 0)
                return count;
    }
}
#endif

return 0;
}

/* Return the total number of processors. Here QUERY must be one of
NPROC_ALL, NPROC_CURRENT. The result is guaranteed to be at least 1. */
static unsigned long int
num_processors_ignoring_omp (enum nproc_query query)
{
/* On systems with a modern affinity mask system call, we have
    sysconf (_SC_NPROCESSORS_CONF)
    >= sysconf (_SC_NPROCESSORS_ONLN)
    >= num_processors_via_affinity_mask ()
The first number is the number of CPUs configured in the system.
The second number is the number of CPUs available to the scheduler.
The third number is the number of CPUs available to the current process.

Note! On Linux systems with glibc, the first and second number come from
the /sys and /proc file systems (see
glibc/sysdeps/unix/sysv/linux/getsysstats.c).
In some situations these file systems are not mounted, and the sysconf call
returns 1 or 2 (<https://sourceware.org/bugzilla/show\_bug.cgi?id=21542>),
which does not reflect the reality. */

if (query == NPROC_CURRENT)
{
/* Try the modern affinity mask system call. */
{
    unsigned long nprocs = num_processors_via_affinity_mask ();

    if (nprocs > 0)
        return nprocs;
}

#ifndef _SC_NPROCESSORS_ONLN
{ /* This works on glibc, Mac OS X 10.5, FreeBSD, AIX, OSF/1, Solaris,
   Cygwin, Haiku. */
    long int nprocs = sysconf (_SC_NPROCESSORS_ONLN);
    if (nprocs > 0)

```

```

        return nprocs;
    }
#endif
}
else /* query == NPROC_ALL */
{
#ifndef _SC_NPROCESSORS_CONF
/* This works on glibc, Mac OS X 10.5, FreeBSD, AIX, OSF/1, Solaris,
   Cygwin, Haiku. */
long int nprocs = sysconf (_SC_NPROCESSORS_CONF);

#if __GLIBC__ >= 2 && defined __linux__
/* On Linux systems with glibc, this information comes from the /sys and
   /proc file systems (see glibc/sysdeps/unix/sysv/linux/getsysstats.c).
   In some situations these file systems are not mounted, and the
   sysconf call returns 1 or 2. But we wish to guarantee that
   num_processors (NPROC_ALL) >= num_processors (NPROC_CURRENT). */
if (nprocs == 1 || nprocs == 2)
{
    unsigned long nprocs_current = num_processors_via_affinity_mask ();

    if /* nprocs_current > 0 && */ nprocs_current > nprocs)
        nprocs = nprocs_current;
}
#endif
#endif
}

if (nprocs > 0)
return nprocs;
}

#endif
}

#endif HAVE_PSTAT_GETDYNAMIC
{ /* This works on HP-UX. */
struct pst_dynamic psd;
if (pststat_getdynamic (&psd, sizeof psd, 1, 0) >= 0)
{
    /* The field psd_proc_cnt contains the number of active processors.
       In newer releases of HP-UX 11, the field psd_max_proc_cnt includes
       deactivated processors. */
    if (query == NPROC_CURRENT)
    {
        if (psd.psd_proc_cnt > 0)
            return psd.psd_proc_cnt;
    }
    else
    {
        if (psd.psd_max_proc_cnt > 0)
            return psd.psd_max_proc_cnt;
    }
}
}

#endif HAVE_SYSMP && defined MP_NAPROCS && defined MP_NPROCS
{ /* This works on IRIX. */
/* MP_NPROCS yields the number of installed processors.
   MP_NAPROCS yields the number of processors available to unprivileged
   processes. */
int nprocs =

```

```

sysmp (query == NPROC_CURRENT && getuid () != 0
    ? MP_NAPROCS
    : MP_NPROCS);
if (nprocs > 0)
    return nprocs;
}
#endif

/* Finally, as fallback, use the APIs that don't distinguish between
   NPROC_CURRENT and NPROC_ALL. */

#if HAVE_SYSCTL && !(defined __GLIBC__ && defined __linux__)
{ /* This works on macOS, FreeBSD, NetBSD, OpenBSD.
   macOS 10.14 does not allow mib to be const. */
    int nprocs;
    size_t len = sizeof (nprocs);
    static int mib[][2] = {
        #ifdef HW_NCPUONLINE
        { CTL_HW, HW_NCPUONLINE },
        #endif
        { CTL_HW, HW_NCPU }
    };
    for (int i = 0; i < ARRAY_SIZE (mib); i++)
    {
        if (sysctl (mib[i], ARRAY_SIZE (mib[i]), &nprocs, &len, NULL, 0) == 0
            && len == sizeof (nprocs)
            && 0 < nprocs)
            return nprocs;
    }
}
#endif

#if defined _WIN32 && ! defined __CYGWIN__
{ /* This works on native Windows platforms. */
    SYSTEM_INFO system_info;
    GetSystemInfo (&system_info);
    if (0 < system_info.dwNumberOfProcessors)
        return system_info.dwNumberOfProcessors;
}
#endif

#if defined(__MVS__)
return __get_num_online_cpus();
#endif

return 1;
}

/* Parse OMP environment variables without dependence on OMP.
Return 0 for invalid values. */
static unsigned long int
parse_omp_threads (char const* threads)
{
    unsigned long int ret = 0;

    if (threads == NULL)
        return ret;

    /* The OpenMP spec says that the value assigned to the environment variables
       "may have leading and trailing white space". */

```

```

while (*threads != '\0' && c_isspace (*threads))
    threads++;

/* Convert it from positive decimal to 'unsigned long'. */
if (c_isdigit (*threads))
{
    char *endptr = NULL;
    unsigned long int value = strtoul (threads, &endptr, 10);

    if (endptr != NULL)
    {
        while (*endptr != '\0' && c_isspace (*endptr))
            endptr++;
        if (*endptr == '\0')
            return value;
        /* Also accept the first value in a nesting level,
           since we can't determine the nesting level from env vars. */
        else if (*endptr == ',')
            return value;
    }
}

return ret;
}

unsigned long int
num_processors (enum nproc_query query)
{
unsigned long int omp_env_limit = ULONG_MAX;

if (query == NPROC_CURRENT_OVERRIDABLE)
{
    unsigned long int omp_env_threads;
    /* Honor the OpenMP environment variables, recognized also by all
       programs that are based on OpenMP. */
    omp_env_threads = parse_omp_threads (getenv ("OMP_NUM_THREADS"));
    omp_env_limit = parse_omp_threads (getenv ("OMP_THREAD_LIMIT"));
    if (!omp_env_limit)
        omp_env_limit = ULONG_MAX;

    if (omp_env_threads)
        return MIN (omp_env_threads, omp_env_limit);

    query = NPROC_CURRENT;
}
/* Here query is one of NPROC_ALL, NPROC_CURRENT. */
{
    unsigned long nprocs = num_processors_ignoring_omp (query);
    return MIN (nprocs, omp_env_limit);
}
"""

,
"patch": """
diff --git a/lib/nproc.c b/lib/nproc.c
index e3de187..b2ab757 100644
--- a/lib/nproc.c
+++ b/lib/nproc.c
@@ -336,6 +336,10 @@ num_processors_ignoring_omp (enum nproc_query query)
"""

```

```

#endif

+if defined(__MVS__)
+ return __get_num_online_cpus();
#endif
+
return 1;
}
"""

},
{
  "wrong_code": """
/* Base64, base32, and similar encoding/decoding strings or files.
Copyright (C) 2004-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */
*/

```

/\* Written by Simon Josefsson <simon@josefsson.org>. \*/

```

#include <config.h>

#include <stdio.h>
#include <getopt.h>
#include <sys/types.h>

#include "system.h"
#include "assure.h"
#include "c-ctype.h"
#include "fadvise.h"
#include "quote.h"
#include "xstrtol.h"
#include "xdectoint.h"
#include "xbinary-io.h"

#if BASE_TYPE == 42
#define AUTHORS \
proper_name ("Simon Josefsson"), \
proper_name ("Assaf Gordon")
#else
#define AUTHORS proper_name ("Simon Josefsson")
#endif

#if BASE_TYPE == 32
#include "base32.h"
#define PROGRAM_NAME "base32"
#elif BASE_TYPE == 64
#include "base64.h"

```

```

#define PROGRAM_NAME "base64"
#elif BASE_TYPE == 42
#include "base32.h"
#include "base64.h"
#include "assure.h"
#define PROGRAM_NAME "basenc"
#else
#error missing/invalid BASE_TYPE definition
#endif

#if BASE_TYPE == 42
enum
{
BASE64_OPTION = CHAR_MAX + 1,
BASE64URL_OPTION,
BASE32_OPTION,
BASE32HEX_OPTION,
BASE16_OPTION,
BASE2MSBF_OPTION,
BASE2LSBF_OPTION,
Z85_OPTION
};
#endif

static struct option const long_options[] =
{
{"decode", no_argument, 0, 'd'},
 {"wrap", required_argument, 0, 'w'},
 {"ignore-garbage", no_argument, 0, 'i'},
#if BASE_TYPE == 42
 {"base64", no_argument, 0, BASE64_OPTION},
 {"base64url", no_argument, 0, BASE64URL_OPTION},
 {"base32", no_argument, 0, BASE32_OPTION},
 {"base32hex", no_argument, 0, BASE32HEX_OPTION},
 {"base16", no_argument, 0, BASE16_OPTION},
 {"base2msbf", no_argument, 0, BASE2MSBF_OPTION},
 {"base2lsbf", no_argument, 0, BASE2LSBF_OPTION},
 {"z85", no_argument, 0, Z85_OPTION},
#endif
{GETOPT_HELP_OPTION_DECL},
{GETOPT_VERSION_OPTION_DECL},
{nullptr, 0, nullptr, 0}
};

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
    {
    printf (_("\
Usage: %s [OPTION]... [FILE]\n\
"), program_name);

#if BASE_TYPE == 42
    fputs (_("\
basenc encode or decode FILE, or standard input, to standard output.\n\

```

```

"), stdout);
#else
    printf(_("\
Base%d encode or decode FILE, or standard input, to standard output.\n\
"), BASE_TYPE);
#endif

    emit_stdin_note ();
    emit_mandatory_arg_note ();
#endif BASE_TYPE == 42
    fputs(_("\
    --base64      same as 'base64' program (RFC4648 section 4)\n\
"), stdout);
    fputs(_("\
    --base64url   file- and url-safe base64 (RFC4648 section 5)\n\
"), stdout);
    fputs(_("\
    --base32      same as 'base32' program (RFC4648 section 6)\n\
"), stdout);
    fputs(_("\
    --base32hex   extended hex alphabet base32 (RFC4648 section 7)\n\
"), stdout);
    fputs(_("\
    --base16      hex encoding (RFC4648 section 8)\n\
"), stdout);
    fputs(_("\
    --base2msbf   bit string with most significant bit (msb) first\n\
"), stdout);
    fputs(_("\
    --base2lsbf   bit string with least significant bit (lsb) first\n\
"), stdout);
#endif
    fputs(_("\
-d, --decode    decode data\n\
-i, --ignore-garbage when decoding, ignore non-alphabet characters\n\
-w, --wrap=COLS wrap encoded lines after COLS character (default 76).\n\
Use 0 to disable line wrapping\n\
"), stdout);
#endif BASE_TYPE == 42
    fputs(_("\
    --z85        ascii85-like encoding (ZeroMQ spec:32/Z85);\n\
when encoding, input length must be a multiple of 4;\n\
when decoding, input length must be a multiple of 5\n\
"), stdout);
#endif
    fputs(HELP_OPTION_DESCRIPTION, stdout);
    fputs(VERSION_OPTION_DESCRIPTION, stdout);
#endif BASE_TYPE == 42
    fputs(_("\
\n\
When decoding, the input may contain newlines in addition to the bytes of\n\
the formal alphabet. Use --ignore-garbage to attempt to recover\n\
from any other non-alphabet bytes in the encoded stream.\n\
"), stdout);
#else
    printf(_("\
\n\
The data are encoded as described for the %s alphabet in RFC 4648.\n\
When decoding, the input may contain newlines in addition to the bytes of\n\
the formal %s alphabet. Use --ignore-garbage to attempt to recover\n\
"), stdout);

```

```

from any other non-alphabet bytes in the encoded stream.\n"),
    PROGRAM_NAME, PROGRAM_NAME);
#endif
    emit_ancillary_info (PROGRAM_NAME);
}

exit (status);
}

#ifndef BASE_TYPE != 64
static int
base32_required_padding (int len)
{
int partial = len % 8;
return partial ? 8 - partial : 0;
}
#endif

#ifndef BASE_TYPE != 32
static int
base64_required_padding (int len)
{
int partial = len % 4;
return partial ? 4 - partial : 0;
}
#endif

#ifndef BASE_TYPE == 42
static int
no_required_padding (int len)
{
return 0;
}
#endif

#define ENC_BLOCKSIZE (1024 * 3 * 10)

#ifndef BASE_TYPE == 32
#define BASE_LENGTH BASE32_LENGTH
#define REQUIRED_PADDING base32_required_padding
/* Note that increasing this may decrease performance if --ignore-garbage
is used, because of the memmove operation below. */
#define DEC_BLOCKSIZE (1024 * 5)

/* Ensure that BLOCKSIZE is a multiple of 5 and 8. */
static_assert (ENC_BLOCKSIZE % 40 == 0); /* Padding chars only on last block. */
static_assert (DEC_BLOCKSIZE % 40 == 0); /* Complete encoded blocks are used. */

#define base_encode base32_encode
#define base_decode_context base32_decode_context
#define base_decode_ctx_init base32_decode_ctx_init
#define base_decode_ctx base32_decode_ctx
#define isubase isubase32
#endif
#ifndef BASE_TYPE == 64
#define BASE_LENGTH BASE64_LENGTH
#define REQUIRED_PADDING base64_required_padding
/* Note that increasing this may decrease performance if --ignore-garbage
is used, because of the memmove operation below. */
#define DEC_BLOCKSIZE (1024 * 3)

```

```

/* Ensure that BLOCKSIZE is a multiple of 3 and 4. */
static_assert (ENC_BLOCKSIZE % 12 == 0); /* Padding chars only on last block. */
static_assert (DEC_BLOCKSIZE % 12 == 0); /* Complete encoded blocks are used. */

#define base_encode base64_encode
#define base_decode_context base64_decode_context
#define base_decode_ctx_init base64_decode_ctx_init
#define base_decode_ctx base64_decode_ctx
#define isubase isubase64
#elif BASE_TYPE == 42

#define BASE_LENGTH base_length
#define REQUIRED_PADDING required_padding

/* Note that increasing this may decrease performance if --ignore-garbage
is used, because of the memmove operation below. */
#define DEC_BLOCKSIZE (4200)
static_assert (DEC_BLOCKSIZE % 40 == 0); /* complete encoded blocks for base32*/
static_assert (DEC_BLOCKSIZE % 12 == 0); /* complete encoded blocks for base64*/

static int (*base_length) (int i);
static int (*required_padding) (int i);
static bool (*isubase) (unsigned char ch);
static void (*base_encode) (char const *restrict in, idx_t inlen,
                           char *restrict out, idx_t outlen);

struct base16_decode_context
{
/* Either a 4-bit nibble, or negative if we have no nibble. */
signed char nibble;
};

struct z85_decode_context
{
int i;
unsigned char octets[5];
};

struct base2_decode_context
{
unsigned char octet;
};

struct base_decode_context
{
int i; /* will be updated manually */
union {
    struct base64_decode_context base64;
    struct base32_decode_context base32;
    struct base16_decode_context base16;
    struct base2_decode_context base2;
    struct z85_decode_context z85;
} ctx;
char *inbuf;
idx_t bufsize;
};
static void (*base_decode_ctx_init) (struct base_decode_context *ctx);
static bool (*base_decode_ctx) (struct base_decode_context *ctx,
                           char const *restrict in, idx_t inlen,
                           char *restrict out, idx_t outlen,
                           idx_t *outlen);

```

```

        char *restrict out, idx_t *outlen);
#endif

#if BASE_TYPE == 42

static int
base64_length_wrapper (int len)
{
return BASE64_LENGTH (len);
}

static void
base64_decode_ctx_init_wrapper (struct base_decode_context *ctx)
{
base64_decode_ctx_init (&ctx->ctx.base64);
}

static bool
base64_decode_ctx_wrapper (struct base_decode_context *ctx,
                           char const *restrict in, idx_t inlen,
                           char *restrict out, idx_t *outlen)
{
bool b = base64_decode_ctx (&ctx->ctx.base64, in, inlen, out, outlen);
ctx->i = ctx->ctx.base64.i;
return b;
}

static void
init_inbuf (struct base_decode_context *ctx)
{
ctx->bufsize = DEC_BLOCKSIZE;
ctx->inbuf = xcharalloc (ctx->bufsize);
}

static void
prepare_inbuf (struct base_decode_context *ctx, idx_t inlen)
{
if (ctx->bufsize < inlen)
{
ctx->bufsize = inlen * 2;
ctx->inbuf = xnrealloc (ctx->inbuf, ctx->bufsize, sizeof (char));
}
}

static void
base64url_encode (char const *restrict in, idx_t inlen,
                  char *restrict out, idx_t outlen)
{
base64_encode (in, inlen, out, outlen);
/* translate 62nd and 63rd characters */
char *p = out;
while (outlen--)
{
if (*p == '+')
    *p = '-';
else if (*p == '/')

```

```

        *p = '_';
        ++p;
    }
}

static bool
isbase64url (unsigned char ch)
{
return (ch == '-' || ch == '_'
        || (ch != '+' && ch != '/' && isbase64 (ch)));
}

static void
base64url_decode_ctx_init_wrapper (struct base_decode_context *ctx)
{
base64_decode_ctx_init (&ctx->ctx.base64);
init_inbuf (ctx);
}

static bool
base64url_decode_ctx_wrapper (struct base_decode_context *ctx,
                           char const *restrict in, idx_t inlen,
                           char *restrict out, idx_t *outlen)
{
prepare_inbuf (ctx, inlen);
memcpy (ctx->inbuf, in, inlen);

/* translate 62nd and 63rd characters */
idx_t i = inlen;
char *p = ctx->inbuf;
while (i--)
{
if (*p == '+' || *p == '/')
{
*outlen = 0;
return false; /* reject base64 input */
}
else if (*p == '-')
*p = '+';
else if (*p == '_')
*p = '/';
++p;
}

bool b = base64_decode_ctx (&ctx->ctx.base64, ctx->inbuf, inlen,
                           out, outlen);
ctx->i = ctx->ctx.base64.i;

return b;
}

static int
base32_length_wrapper (int len)
{
return BASE32_LENGTH (len);
}

```

```

static void
base32_decode_ctx_init_wrapper (struct base_decode_context *ctx)
{
base32_decode_ctx_init (&ctx->ctx.base32);
}

static bool
base32_decode_ctx_wrapper (struct base_decode_context *ctx,
                           char const *restrict in, idx_t inlen,
                           char *restrict out, idx_t *outlen)
{
bool b = base32_decode_ctx (&ctx->ctx.base32, in, inlen, out, outlen);
ctx->i = ctx->ctx.base32.i;
return b;
}

/* ABCDEFGHIJKLMNOPQRSTUVWXYZ234567
   to
0123456789ABCDEFGHIJKLMNOPQRSTU */
static const char base32_norm_to_hex[32 + 9] = {
/*0x32, 0x33, 0x34, 0x35, 0x36, 0x37, */
'Q', 'R', 'S', 'T', 'U', 'V',
0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f, 0x40,
/*0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, */
'0', '1', '2', '3', '4', '5', '6', '7',
/*0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f, 0x50, */
'8', '9', 'A', 'B', 'C', 'D', 'E', 'F',
/*0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, */
'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
/*0x59, 0x5a, */
'O', 'P',
};

/* 0123456789ABCDEFGHIJKLMNOPQRSTU */
/* to
ABCDEFGHIJKLMNOPQRSTUVWXYZ234567 */
static const char base32_hex_to_norm[32 + 9] = {
/* from: 0x30 .. 0x39 ('0' to '9') */
/* to: */ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f, 0x40,
/* from: 0x41 .. 0x4A ('A' to 'J') */
/* to: */ 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
/* from: 0x4B .. 0x54 ('K' to 'T') */
/* to: */ 'U', 'V', 'W', 'X', 'Y', 'Z', '2', '3', '4', '5',
/* from: 0x55 .. 0x56 ('U' to 'V') */
/* to: */ '6', '7'
};

inline static bool
isubase32hex (unsigned char ch)

```

```

{
    return ('0' <= ch && ch <= '9') || ('A' <= ch && ch <= 'V');
}

static void
base32hex_encode (char const *restrict in, idx_t inlen,
                  char *restrict out, idx_t outlen)
{
    base32_encode (in, inlen, out, outlen);

    for (char *p = out; outlen--; p++)
    {
        affirm (0x32 <= *p && *p <= 0x5a); /* LCOV_EXCL_LINE */
        *p = base32_norm_to_hex[*p - 0x32];
    }
}

static void
base32hex_decode_ctx_init_wrapper (struct base_decode_context *ctx)
{
    base32_decode_ctx_init (&ctx->ctx.base32);
    init_inbuf (ctx);
}

static bool
base32hex_decode_ctx_wrapper (struct base_decode_context *ctx,
                             char const *restrict in, idx_t inlen,
                             char *restrict out, idx_t *outlen)
{
    prepare_inbuf (ctx, inlen);

    idx_t i = inlen;
    char *p = ctx->inbuf;
    while (i--)
    {
        if (isubase32hex (*in))
            *p = base32_hex_to_norm[*in - 0x30];
        else
            *p = *in;
        ++p;
        ++in;
    }

    bool b = base32_decode_ctx (&ctx->ctx.base32, ctx->inbuf, inlen,
                               out, outlen);
    ctx->i = ctx->ctx.base32.i;

    return b;
}
/* With this approach this file works independent of the charset used
(think EBCDIC). However, it does assume that the characters in the
Base32 alphabet (A-Z2-7) are encoded in 0..255. POSIX
1003.1-2001 require that char and unsigned char are 8-bit
quantities, though, taking care of that problem. But this may be a
potential problem on non-POSIX C99 platforms.

```

IBM C V6 for AIX mishandles "#define B32(x) ...'x'...", so use "\_"

```

as the formal parameter rather than "x". */
#define B16(_)
((_) == '0' ? 0 \
: (_ == '1' ? 1 \
: (_ == '2' ? 2 \
: (_ == '3' ? 3 \
: (_ == '4' ? 4 \
: (_ == '5' ? 5 \
: (_ == '6' ? 6 \
: (_ == '7' ? 7 \
: (_ == '8' ? 8 \
: (_ == '9' ? 9 \
: (_ == 'A' || (_ == 'a' ? 10 \
: (_ == 'B' || (_ == 'b' ? 11 \
: (_ == 'C' || (_ == 'c' ? 12 \
: (_ == 'D' || (_ == 'd' ? 13 \
: (_ == 'E' || (_ == 'e' ? 14 \
: (_ == 'F' || (_ == 'f' ? 15 \
: -1)

```

```

static signed char const base16_to_int[256] = {
B16 (0), B16 (1), B16 (2), B16 (3),
B16 (4), B16 (5), B16 (6), B16 (7),
B16 (8), B16 (9), B16 (10), B16 (11),
B16 (12), B16 (13), B16 (14), B16 (15),
B16 (16), B16 (17), B16 (18), B16 (19),
B16 (20), B16 (21), B16 (22), B16 (23),
B16 (24), B16 (25), B16 (26), B16 (27),
B16 (28), B16 (29), B16 (30), B16 (31),
B16 (32), B16 (33), B16 (34), B16 (35),
B16 (36), B16 (37), B16 (38), B16 (39),
B16 (40), B16 (41), B16 (42), B16 (43),
B16 (44), B16 (45), B16 (46), B16 (47),
B16 (48), B16 (49), B16 (50), B16 (51),
B16 (52), B16 (53), B16 (54), B16 (55),
B16 (56), B16 (57), B16 (58), B16 (59),
B16 (60), B16 (61), B16 (62), B16 (63),
B16 (64), B16 (65), B16 (66), B16 (67),
B16 (68), B16 (69), B16 (70), B16 (71),
B16 (72), B16 (73), B16 (74), B16 (75),
B16 (76), B16 (77), B16 (78), B16 (79),
B16 (80), B16 (81), B16 (82), B16 (83),
B16 (84), B16 (85), B16 (86), B16 (87),
B16 (88), B16 (89), B16 (90), B16 (91),
B16 (92), B16 (93), B16 (94), B16 (95),
B16 (96), B16 (97), B16 (98), B16 (99),
B16 (100), B16 (101), B16 (102), B16 (103),
B16 (104), B16 (105), B16 (106), B16 (107),
B16 (108), B16 (109), B16 (110), B16 (111),
B16 (112), B16 (113), B16 (114), B16 (115),
B16 (116), B16 (117), B16 (118), B16 (119),
B16 (120), B16 (121), B16 (122), B16 (123),
B16 (124), B16 (125), B16 (126), B16 (127),
B16 (128), B16 (129), B16 (130), B16 (131),
B16 (132), B16 (133), B16 (134), B16 (135),
B16 (136), B16 (137), B16 (138), B16 (139),
B16 (140), B16 (141), B16 (142), B16 (143),
B16 (144), B16 (145), B16 (146), B16 (147),
B16 (148), B16 (149), B16 (150), B16 (151),
B16 (152), B16 (153), B16 (154), B16 (155),

```

```

B16 (156), B16 (157), B16 (158), B16 (159),
B16 (160), B16 (161), B16 (162), B16 (163),
B16 (132), B16 (165), B16 (166), B16 (167),
B16 (168), B16 (169), B16 (170), B16 (171),
B16 (172), B16 (173), B16 (174), B16 (175),
B16 (176), B16 (177), B16 (178), B16 (179),
B16 (180), B16 (181), B16 (182), B16 (183),
B16 (184), B16 (185), B16 (186), B16 (187),
B16 (188), B16 (189), B16 (190), B16 (191),
B16 (192), B16 (193), B16 (194), B16 (195),
B16 (196), B16 (197), B16 (198), B16 (199),
B16 (200), B16 (201), B16 (202), B16 (203),
B16 (204), B16 (205), B16 (206), B16 (207),
B16 (208), B16 (209), B16 (210), B16 (211),
B16 (212), B16 (213), B16 (214), B16 (215),
B16 (216), B16 (217), B16 (218), B16 (219),
B16 (220), B16 (221), B16 (222), B16 (223),
B16 (224), B16 (225), B16 (226), B16 (227),
B16 (228), B16 (229), B16 (230), B16 (231),
B16 (232), B16 (233), B16 (234), B16 (235),
B16 (236), B16 (237), B16 (238), B16 (239),
B16 (240), B16 (241), B16 (242), B16 (243),
B16 (244), B16 (245), B16 (246), B16 (247),
B16 (248), B16 (249), B16 (250), B16 (251),
B16 (252), B16 (253), B16 (254), B16 (255)
};

static bool
isbase16 (unsigned char ch)
{
return ch < sizeof base16_to_int && 0 <= base16_to_int[ch];
}

static int
base16_length (int len)
{
return len * 2;
}

static void
base16_encode (char const *restrict in, idx_t inlen,
               char *restrict out, idx_t outlen)
{
static const char base16[16] = "0123456789ABCDEF";

while (inlen && outlen)
{
    unsigned char c = *in;
    *out++ = base16[c >> 4];
    *out++ = base16[c & 0x0F];
    ++in;
    inlen--;
    outlen -= 2;
}
}

static void
base16_decode_ctx_init (struct base_decode_context *ctx)

```

```

{
init_inbuf (ctx);
ctx->ctx.base16.nibble = -1;
ctx->i = 1;
}

static bool
base16_decode_ctx (struct base_decode_context *ctx,
                  char const *restrict in, idx_t inlen,
                  char *restrict out, idx_t *outlen)
{
bool ignore_lines = true; /* for now, always ignore them */
char *out0 = out;
signed char nibble = ctx->ctx.base16.nibble;

/* inlen==0 is request to flush output.
   if there is a dangling high nibble - we are missing the low nibble,
   so return false - indicating an invalid input. */
if (inlen == 0)
{
    *
    *outlen = 0;
    return nibble < 0;
}

while (inlen--)
{
    *
    unsigned char c = *in++;
    if (ignore_lines && c == '\n')
        continue;

    if (sizeof base16_to_int <= c || base16_to_int[c] < 0)
    {
        *
        *outlen = out - out0;
        return false; /* garbage - return false */
    }

    if (nibble < 0)
        nibble = base16_to_int[c];
    else
    {
        /* have both nibbles, write octet */
        *out++ = (nibble << 4) + base16_to_int[c];
        nibble = -1;
    }
}

ctx->ctx.base16.nibble = nibble;
*outlen = out - out0;
return true;
}

```

```

static int
z85_length (int len)
{
/* Z85 does not allow padding, so no need to round to highest integer. */
int outlen = (len * 5) / 4;

```

```

return outlen;
}

static bool
isuz85 (unsigned char ch)
{
return c_isalnum (ch) || strchr (".-:+=^!/*?&<>()[]{}@%$#", ch) != nullptr;
}

static char const z85_encoding[85] =
"0123456789"
"abcdefghijklmnopqrstuvwxyz"
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
".-:+=^!/*?&<>()[]{}@%$#";

static void
z85_encode (char const *restrict in, idx_t inlen,
            char *restrict out, idx_t outlen)
{
int i = 0;
unsigned char quad[4];
idx_t outidx = 0;

while (true)
{
if (inlen == 0)
{
/* no more input, exactly on 4 octet boundary. */
if (i == 0)
return;

/* currently, there's no way to return an error in encoding. */
error (EXIT_FAILURE, 0,
      _("invalid input (length must be multiple of 4 characters")));
}
else
{
quad[i++] = *in++;
--inlen;
}

/* Got a quad, encode it */
if (i == 4)
{
int_fast64_t val = quad[0];
val = (val << 24) + (quad[1] << 16) + (quad[2] << 8) + quad[3];

for (int j = 4; j >= 0; --j)
{
int c = val % 85;
val /= 85;

/* NOTE: if there is padding (which is trimmed by z85
before outputting the result), the output buffer 'out'
might not include enough allocated bytes for the padding,
so don't store them. */
if (outidx + j < outlen)
out[j] = z85_encoding[c];
}
out += 5;
}
}

```

```

        outidx += 5;
        i = 0;
    }
}

static void
z85_decode_ctx_init (struct base_decode_context *ctx)
{
init_inbuf (ctx);
ctx->ctx.z85.i = 0;
ctx->i = 1;
}

#define Z85_LO_CTX_TO_32BIT_VAL(ctx) \
(((ctx)->ctx.z85.octets[1] * 85 * 85 * 85) + \
((ctx)->ctx.z85.octets[2] * 85 * 85) + \
((ctx)->ctx.z85.octets[3] * 85) + \
((ctx)->ctx.z85.octets[4]))

#define Z85_HI_CTX_TO_32BIT_VAL(ctx) \
((int_fast64_t) (ctx)->ctx.z85.octets[0] * 85 * 85 * 85 * 85)

/*
0 - 9: 0 1 2 3 4 5 6 7 8 9
10 - 19: a b c d e f g h i j
20 - 29: k l m n o p q r s t
30 - 39: u v w x y z A B C D
40 - 49: E F G H I J K L M N
50 - 59: O P Q R S T U V W X
60 - 69: Y Z . - : + = ^ ! / #dummy comment to workaround syntax-check
70 - 79: * ? & < > () [] {
80 - 84: } @ % $ #
*/
static signed char const z85_decoding[93] = {
68, -1, 84, 83, 82, 72, -1, /* ! " # $ % & ' */
75, 76, 70, 65, -1, 63, 62, 69, /* ( ) * + , - . / */
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, /* '0' to '9' */
64, -1, 73, 66, 74, 71, 81, /* : ; < = > ? @ */
36, 37, 38, 39, 40, 41, 42, 43, 44, 45, /* 'A' to 'J' */
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, /* 'K' to 'T' */
56, 57, 58, 59, 60, 61, /* 'U' to 'Z' */
77, -1, 78, 67, -1, -1, /* [ \ ] ^ _ ` */
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, /* 'a' to 'j' */
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, /* 'k' to 't' */
30, 31, 32, 33, 34, 35, /* 'u' to 'z' */
79, -1, 80 /* { | } */
};

static bool
z85_decode_ctx (struct base_decode_context *ctx,
                char const *restrict in, idx_t inlen,
                char *restrict out, idx_t *outlen)
{
bool ignore_lines = true; /* for now, always ignore them */

*outlen = 0;

```

```

/* inlen==0 is request to flush output.
   if there are dangling values - we are missing entries,
   so return false - indicating an invalid input. */
if (inlen == 0)
{
    if (ctx->ctx.z85.i > 0)
    {
        /* Z85 variant does not allow padding - input must
           be a multiple of 5 - so return error. */
        return false;
    }
    return true;
}

while (inlen--)
{
    if (ignore_lines && *in == '\n')
    {
        ++in;
        continue;
    }

/* z85 decoding */
unsigned char c = *in;

if (c >= 33 && c <= 125)
{
    signed char ch = z85_decoding[c - 33];
    if (ch < 0)
        return false; /* garbage - return false */
    c = ch;
}
else
    return false; /* garbage - return false */

++in;

ctx->ctx.z85.octets[ctx->ctx.z85.i++] = c;
if (ctx->ctx.z85.i == 5)
{
    /* decode the lowest 4 octets, then check for overflows. */
    int_fast64_t val = Z85_LO_CTX_TO_32BIT_VAL(ctx);

    /* The Z85 spec and the reference implementation say nothing
       about overflows. To be on the safe side, reject them. */

    val += Z85_HI_CTX_TO_32BIT_VAL(ctx);
    if ((val >> 24) & ~0xFF)
        return false;

    *out++ = val >> 24;
    *out++ = (val >> 16) & 0xFF;
    *out++ = (val >> 8) & 0xFF;
    *out++ = val & 0xFF;

    *outlen += 4;

    ctx->ctx.z85.i = 0;
}
}

```

```

ctx->i = ctx->ctx.z85.i;
return true;
}

inline static bool
isubase2 (unsigned char ch)
{
return ch == '0' || ch == '1';
}

static int
base2_length (int len)
{
return len * 8;
}

inline static void
base2msbf_encode (char const *restrict in, idx_t inlen,
                  char *restrict out, idx_t outlen)
{
while (inlen && outlen)
{
    unsigned char c = *in;
    for (int i = 0; i < 8; i++)
    {
        *out++ = c & 0x80 ? '1' : '0';
        c <= 1;
    }
    inlen--;
    outlen -= 8;
    ++in;
}
}

inline static void
base2lsbf_encode (char const *restrict in, idx_t inlen,
                  char *restrict out, idx_t outlen)
{
while (inlen && outlen)
{
    unsigned char c = *in;
    for (int i = 0; i < 8; i++)
    {
        *out++ = c & 0x01 ? '1' : '0';
        c >= 1;
    }
    inlen--;
    outlen -= 8;
    ++in;
}
}

static void
base2_decode_ctx_init (struct base_decode_context *ctx)
{
init_inbuf (ctx);
ctx->ctx.base2.octet = 0;
}

```

```

ctx->i = 0;
}

static bool
base2lsbf_decode_ctx (struct base_decode_context *ctx,
                      char const *restrict in, idx_t inlen,
                      char *restrict out, idx_t *outlen)
{
bool ignore_lines = true; /* for now, always ignore them */

*outlen = 0;

/* inlen==0 is request to flush output.
   if there is a dangling bit - we are missing some bits,
   so return false - indicating an invalid input. */
if (inlen == 0)
    return ctx->i == 0;

while (inlen--)
{
    if (ignore_lines && *in == '\n')
    {
        ++in;
        continue;
    }

    if (!isbase2 (*in))
        return false;

    bool bit = (*in == '1');
    ctx->ctx.base2.octet |= bit << ctx->i;
    ++ctx->i;

    if (ctx->i == 8)
    {
        *out++ = ctx->ctx.base2.octet;
        ctx->ctx.base2.octet = 0;
        ++*outlen;
        ctx->i = 0;
    }

    ++in;
}

return true;
}

static bool
base2msbf_decode_ctx (struct base_decode_context *ctx,
                      char const *restrict in, idx_t inlen,
                      char *restrict out, idx_t *outlen)
{
bool ignore_lines = true; /* for now, always ignore them */

*outlen = 0;

/* inlen==0 is request to flush output.
   if there is a dangling bit - we are missing some bits,
   so return false - indicating an invalid input. */

```

```

if (inlen == 0)
    return ctx->i == 0;

while (inlen--)
{
    if (ignore_lines && *in == '\n')
    {
        ++in;
        continue;
    }

    if (!isubbase2 (*in))
        return false;

    bool bit = (*in == '1');
    if (ctx->i == 0)
        ctx->i = 8;
    --ctx->i;
    ctx->ctx.base2.octet |= bit << ctx->i;

    if (ctx->i == 0)
    {
        *out++ = ctx->ctx.base2.octet;
        ctx->ctx.base2.octet = 0;
        ++*outlen;
        ctx->i = 0;
    }

    ++in;
}

return true;
}

#endif /* BASE_TYPE == 42, i.e., "basenc"*/


static void
wrap_write (char const *buffer, idx_t len,
            idx_t wrap_column, idx_t *current_column, FILE *out)
{
if (wrap_column == 0)
{
    /* Simple write. */
    if (fwrite (buffer, 1, len, stdout) < len)
        write_error ();
}
else
for (idx_t written = 0; written < len; )
{
    idx_t to_write = MIN (wrap_column - *current_column, len - written);

    if (to_write == 0)
    {
        if (fputc ('\n', out) == EOF)
            write_error ();
        *current_column = 0;
    }
    else

```

```

    {
        if (fwrite (buffer + written, 1, to_write, stdout) < to_write)
            write_error ();
        *current_column += to_write;
        written += to_write;
    }
}

static __Noreturn void
finish_and_exit (FILE *in, char const *infile)
{
if (fclose (in) != 0)
{
    if (STREQ (infile, "-"))
        error (EXIT_FAILURE, errno, _("closing standard input"));
    else
        error (EXIT_FAILURE, errno, "%s", quotef (infile));
}

exit (EXIT_SUCCESS);
}

static __Noreturn void
do_encode (FILE *in, char const *infile, FILE *out, idx_t wrap_column)
{
idx_t current_column = 0;
char *inbuf, *outbuf;
idx_t sum;

inbuf = xmalloc (ENC_BLOCKSIZE);
outbuf = xmalloc (BASE_LENGTH (ENC_BLOCKSIZE));

do
{
    idx_t n;

    sum = 0;
    do
    {
        n = fread (inbuf + sum, 1, ENC_BLOCKSIZE - sum, in);
        sum += n;
    }
    while (!feof (in) && !ferror (in) && sum < ENC_BLOCKSIZE);

    if (sum > 0)
    {
        /* Process input one block at a time. Note that ENC_BLOCKSIZE
         * is sized so that no pad chars will appear in output. */
        base_encode (inbuf, sum, outbuf, BASE_LENGTH (sum));

        wrap_write (outbuf, BASE_LENGTH (sum), wrap_column,
                    &current_column, out);
    }
}
while (!feof (in) && !ferror (in) && sum == ENC_BLOCKSIZE);

/* When wrapping, terminate last line. */
if (wrap_column && current_column > 0 && fputc ('\n', out) == EOF)
    write_error ();

```

```

if (ferror (in))
    error (EXIT_FAILURE, errno, _("read error"));

finish_and_exit (in, infile);
}

static _Noreturn void
do_decode (FILE *in, char const *infile, FILE *out, bool ignore_garbage)
{
char *inbuf, *outbuf;
idx_t sum;
struct base_decode_context ctx;

char padbuf[8] = "=====";
inbuf = xmalloc (BASE_LENGTH (DEC_BLOCKSIZE));
outbuf = xmalloc (DEC_BLOCKSIZE);

#if BASE_TYPE == 42
ctx.inbuf = nullptr;
#endif
base_decode_ctx_init (&ctx);

do
{
    bool ok;

    sum = 0;
    do
    {
        idx_t n = fread (inbuf + sum,
                         1, BASE_LENGTH (DEC_BLOCKSIZE) - sum, in);

        if (ignore_garbage)
        {
            for (idx_t i = 0; n > 0 && i < n;)
            {
                if (isbase (inbuf[sum + i]) || inbuf[sum + i] == '=')
                    i++;
                else
                    memmove (inbuf + sum + i, inbuf + sum + i + 1, --n - i);
            }
        }

        sum += n;

        if (ferror (in))
            error (EXIT_FAILURE, errno, _("read error"));
    }
    while (sum < BASE_LENGTH (DEC_BLOCKSIZE) && !feof (in));

/* The following "loop" is usually iterated just once.
   However, when it processes the final input buffer, we want
   to iterate it one additional time, but with an indicator
   telling it to flush what is in CTX. */
for (int k = 0; k < 1 + !!feof (in); k++)
{
    if (k == 1)
    {
        if (ctx.i == 0)

```

```

        break;

    /* auto pad input (at eof). */
    idx_t auto_padding = REQUIRED_PADDING (ctx.i);
    if (auto_padding && (sum == 0 || inbuf[sum - 1] != '='))
    {
        affirm (auto_padding <= sizeof (padbuf));
        IF_LINT (free (inbuf));
        sum = auto_padding;
        inbuf = padbuf;
    }
    else
        sum = 0; /* process ctx buffer only */
    }
    idx_t n = DEC_BLOCKSIZE;
    ok = base_decode_ctx (&ctx, inbuf, sum, outbuf, &n);

    if (fwrite (outbuf, 1, n, out) < n)
        write_error ();

    if (!ok)
        error (EXIT_FAILURE, 0, _("invalid input"));
    }
}
while (!feof (in));

finish_and_exit (in, infile);
}

int
main (int argc, char **argv)
{
int opt;
FILE *input_fh;
char const *infile;

/* True if --decode has been given and we should decode data. */
bool decode = false;
/* True if we should ignore non-base-alphabetic characters. */
bool ignore_garbage = false;
/* Wrap encoded data around the 76th column, by default. */
idx_t wrap_column = 76;

#if BASE_TYPE == 42
int base_type = 0;
#endif

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

atexit (close_stdout);

while ((opt = getopt_long (argc, argv, "diw:", long_options, nullptr)) != -1)
    switch (opt)
    {
    case 'd':
        decode = true;

```

```

        break;

    case 'w':
    {
        intmax_t w;
        strtol_error s_err = xstrtoimax (optarg, nullptr, 10, &w, "");
        if (LONGINT_OVERFLOW < s_err || w < 0)
            error (EXIT_FAILURE, 0, "%s: %s",
                   _("invalid wrap size"), quote (optarg));
        wrap_column = s_err == LONGINT_OVERFLOW || IDX_MAX < w ? 0 : w;
    }
    break;

    case 'i':
        ignore_garbage = true;
    break;

#ifndef BASE_TYPE == 42
    case BASE64_OPTION:
    case BASE64URL_OPTION:
    case BASE32_OPTION:
    case BASE32HEX_OPTION:
    case BASE16_OPTION:
    case BASE2MSBF_OPTION:
    case BASE2LSBF_OPTION:
    case Z85_OPTION:
        base_type = opt;
        break;
#endif

    case_GETOPT_HELP_CHAR;

    case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

    default:
        usage (EXIT_FAILURE);
        break;
}

#ifndef BASE_TYPE == 42
switch (base_type)
{
    case BASE64_OPTION:
        base_length = base64_length_wrapper;
        required_padding = base64_required_padding;
        isubase = isubase64;
        base_encode = base64_encode;
        base_decode_ctx_init = base64_decode_ctx_init_wrapper;
        base_decode_ctx = base64_decode_ctx_wrapper;
        break;

    case BASE64URL_OPTION:
        base_length = base64_length_wrapper;
        required_padding = base64_required_padding;
        isubase = isubase64url;
        base_encode = base64url_encode;
        base_decode_ctx_init = base64url_decode_ctx_init_wrapper;
        base_decode_ctx = base64url_decode_ctx_wrapper;
        break;
}

```

```

case BASE32_OPTION:
base_length = base32_length_wrapper;
required_padding = base32_required_padding;
isubase = isubase32;
base_encode = base32_encode;
base_decode_ctx_init = base32_decode_ctx_init_wrapper;
base_decode_ctx = base32_decode_ctx_wrapper;
break;

case BASE32HEX_OPTION:
base_length = base32_length_wrapper;
required_padding = base32_required_padding;
isubase = isubase32hex;
base_encode = base32hex_encode;
base_decode_ctx_init = base32hex_decode_ctx_init_wrapper;
base_decode_ctx = base32hex_decode_ctx_wrapper;
break;

case BASE16_OPTION:
base_length = base16_length;
required_padding = no_required_padding;
isubase = isubase16;
base_encode = base16_encode;
base_decode_ctx_init = base16_decode_ctx_init;
base_decode_ctx = base16_decode_ctx;
break;

case BASE2MSBF_OPTION:
base_length = base2_length;
required_padding = no_required_padding;
isubase = isubase2;
base_encode = base2msbf_encode;
base_decode_ctx_init = base2_decode_ctx_init;
base_decode_ctx = base2msbf_decode_ctx;
break;

case BASE2LSBF_OPTION:
base_length = base2_length;
required_padding = no_required_padding;
isubase = isubase2;
base_encode = base2lsbf_encode;
base_decode_ctx_init = base2_decode_ctx_init;
base_decode_ctx = base2lsbf_decode_ctx;
break;

case Z85_OPTION:
base_length = z85_length;
required_padding = no_required_padding;
isubase = isuz85;
base_encode = z85_encode;
base_decode_ctx_init = z85_decode_ctx_init;
base_decode_ctx = z85_decode_ctx;
break;

default:
error (0, 0, _("missing encoding type"));
usage (EXIT_FAILURE);
}

#endif

```

```

if (argc - optind > 1)
{
    error (0, 0, _("extra operand %s"), quote (argv[optind + 1]));
    usage (EXIT_FAILURE);
}

if (optind < argc)
    infile = argv[optind];
else
    infile = "-";

if (STREQ (infile, "-"))
{
    xset_binary_mode (STDIN_FILENO, O_BINARY);
    input_fh = stdin;
}
else
{
    input_fh = fopen (infile, "rb");
    if (input_fh == nullptr)
        error (EXIT_FAILURE, errno, "%s", quotef (infile));
}
fadvise (input_fh, FADVISE_SEQUENTIAL);

if (decode)
    do_decode (input_fh, infile, stdout, ignore_garbage);
else
    do_encode (input_fh, infile, stdout, wrap_column);
}
""",
"error_category": "Platform-Specific Runtime Error",
"error": "CEE3204S The system detected a protection exception (System Completion Code=0C4).",
"correct_code": """
/* Base64, base32, and similar encoding/decoding strings or files.
Copyright (C) 2004-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */

/* Written by Simon Josefsson <simon@josefsson.org>. */

#include <config.h>

#include <stdio.h>
#include <getopt.h>
#include <sys/types.h>
```

```

#include "system.h"
#include "assure.h"
#include "c-ctype.h"
#include "fadvise.h"
#include "quote.h"
#include "xstrtol.h"
#include "xdectoint.h"
#include "xbinary-io.h"

#if BASE_TYPE == 42
# define AUTHORS \
proper_name ("Simon Josefsson"), \
proper_name ("Assaf Gordon")
#else
# define AUTHORS proper_name ("Simon Josefsson")
#endif

#if BASE_TYPE == 32
# include "base32.h"
# define PROGRAM_NAME "base32"
#elif BASE_TYPE == 64
# include "base64.h"
# define PROGRAM_NAME "base64"
#elif BASE_TYPE == 42
# include "base32.h"
# include "base64.h"
# include "assure.h"
# define PROGRAM_NAME "basenc"
#else
# error missing/invalid BASE_TYPE definition
#endif

```

```

#if BASE_TYPE == 42
enum
{
BASE64_OPTION = CHAR_MAX + 1,
BASE64URL_OPTION,
BASE32_OPTION,
BASE32HEX_OPTION,
BASE16_OPTION,
BASE2MSBF_OPTION,
BASE2LSBF_OPTION,
Z85_OPTION
};
#endif

static struct option const long_options[] =
{
{"decode", no_argument, 0, 'd'},
 {"wrap", required_argument, 0, 'w'},
 {"ignore-garbage", no_argument, 0, 'i'},
#if BASE_TYPE == 42
 {"base64", no_argument, 0, BASE64_OPTION},
 {"base64url", no_argument, 0, BASE64URL_OPTION},
 {"base32", no_argument, 0, BASE32_OPTION},
 {"base32hex", no_argument, 0, BASE32HEX_OPTION},
 {"base16", no_argument, 0, BASE16_OPTION},
 {"base2msbf", no_argument, 0, BASE2MSBF_OPTION},

```

```

{"base2lsbf", no_argument, 0, BASE2LSBF_OPTION},
 {"z85",      no_argument, 0, Z85_OPTION},
 #endif
 {GETOPT_HELP_OPTION_DECL},
 {GETOPT_VERSION_OPTION_DECL},
 {nullptr, 0, nullptr, 0}
};

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    printf (_("\
Usage: %s [OPTION]... [FILE]\n\
"), program_name);

#if BASE_TYPE == 42
    fputs (_("\
basenc encode or decode FILE, or standard input, to standard output.\n\
"), stdout);
#else
    printf (_("\
Base%d encode or decode FILE, or standard input, to standard output.\n\
"), BASE_TYPE);
#endif

    emit_stdin_note ();
    emit_mandatory_arg_note ();
#endif
#if BASE_TYPE == 42
    fputs (_("\
--base64      same as 'base64' program (RFC4648 section 4)\n\
"), stdout);
    fputs (_("\
--base64url   file- and url-safe base64 (RFC4648 section 5)\n\
"), stdout);
    fputs (_("\
--base32      same as 'base32' program (RFC4648 section 6)\n\
"), stdout);
    fputs (_("\
--base32hex   extended hex alphabet base32 (RFC4648 section 7)\n\
"), stdout);
    fputs (_("\
--base16      hex encoding (RFC4648 section 8)\n\
"), stdout);
    fputs (_("\
--base2msbf   bit string with most significant bit (msb) first\n\
"), stdout);
    fputs (_("\
--base2lsbf   bit string with least significant bit (lsb) first\n\
"), stdout);
#endif
    fputs (_("\
-d, --decode   decode data\n\
-i, --ignore-garbage when decoding, ignore non-alphabet characters\n\
-w, --wrap=COLS wrap encoded lines after COLS character (default 76).\n\
                  Use 0 to disable line wrapping\n\
"), stdout);

```

```

#if BASE_TYPE == 42
    fputs (_("\
--z85      ascii85-like encoding (ZeroMQ spec:32/Z85);\n\
                when encoding, input length must be a multiple of 4;\n\
                when decoding, input length must be a multiple of 5\n\
"), stdout);
#endif
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
#endif BASE_TYPE == 42
    fputs (_("\
\n\
When decoding, the input may contain newlines in addition to the bytes of\n\
the formal alphabet. Use --ignore-garbage to attempt to recover\n\
from any other non-alphabet bytes in the encoded stream.\n\
"), stdout);
#else
    printf (_("\
\n\
The data are encoded as described for the %s alphabet in RFC 4648.\n\
When decoding, the input may contain newlines in addition to the bytes of\n\
the formal %s alphabet. Use --ignore-garbage to attempt to recover\n\
from any other non-alphabet bytes in the encoded stream.\n"),
        PROGRAM_NAME, PROGRAM_NAME);
#endif
    emit_ancillary_info (PROGRAM_NAME);
}

exit (status);
}

#endif BASE_TYPE != 64
static int
base32_required_padding (int len)
{
int partial = len % 8;
return partial ? 8 - partial : 0;
}
#endif

#endif BASE_TYPE != 32
static int
base64_required_padding (int len)
{
int partial = len % 4;
return partial ? 4 - partial : 0;
}
#endif

#endif BASE_TYPE == 42
static int
no_required_padding (int len)
{
return 0;
}
#endif

#define ENC_BLOCKSIZE (1024 * 3 * 10)

#endif BASE_TYPE == 32

```

```

#define BASE_LENGTH BASE32_LENGTH
#define REQUIRED_PADDING base32_required_padding
/* Note that increasing this may decrease performance if --ignore-garbage
is used, because of the memmove operation below. */
#define DEC_BLOCKSIZE (1024 * 5)

/* Ensure that BLOCKSIZE is a multiple of 5 and 8. */
static_assert (ENC_BLOCKSIZE % 40 == 0); /* Padding chars only on last block. */
static_assert (DEC_BLOCKSIZE % 40 == 0); /* Complete encoded blocks are used. */

#define base_encode base32_encode
#define base_decode_context base32_decode_context
#define base_decode_ctx_init base32_decode_ctx_init
#define base_decode_ctx base32_decode_ctx
#define isubase isubase32
#elif BASE_TYPE == 64
#define BASE_LENGTH BASE64_LENGTH
#define REQUIRED_PADDING base64_required_padding
/* Note that increasing this may decrease performance if --ignore-garbage
is used, because of the memmove operation below. */
#define DEC_BLOCKSIZE (1024 * 3)

/* Ensure that BLOCKSIZE is a multiple of 3 and 4. */
static_assert (ENC_BLOCKSIZE % 12 == 0); /* Padding chars only on last block. */
static_assert (DEC_BLOCKSIZE % 12 == 0); /* Complete encoded blocks are used. */

#define base_encode base64_encode
#define base_decode_context base64_decode_context
#define base_decode_ctx_init base64_decode_ctx_init
#define base_decode_ctx base64_decode_ctx
#define isubase isubase64
#elif BASE_TYPE == 42

#define BASE_LENGTH base_length
#define REQUIRED_PADDING required_padding

/* Note that increasing this may decrease performance if --ignore-garbage
is used, because of the memmove operation below. */
#define DEC_BLOCKSIZE (4200)
static_assert (DEC_BLOCKSIZE % 40 == 0); /* complete encoded blocks for base32*/
static_assert (DEC_BLOCKSIZE % 12 == 0); /* complete encoded blocks for base64*/

static int (*base_length) (int i);
static int (*required_padding) (int i);
static bool (*isubase) (unsigned char ch);
static void (*base_encode) (char const *restrict in, idx_t inlen,
                           char *restrict out, idx_t outlen);

struct base16_decode_context
{
    /* Either a 4-bit nibble, or negative if we have no nibble. */
    signed char nibble;
};

struct z85_decode_context
{
    int i;
    unsigned char octets[5];
};

```

```

struct base2_decode_context
{
    unsigned char octet;
};

struct base_decode_context
{
    int i; /* will be updated manually */
    union {
        struct base64_decode_context base64;
        struct base32_decode_context base32;
        struct base16_decode_context base16;
        struct base2_decode_context base2;
        struct z85_decode_context z85;
    } ctx;
    char *inbuf;
    idx_t bufsize;
};
static void (*base_decode_ctx_init) (struct base_decode_context *ctx);
static bool (*base_decode_ctx) (struct base_decode_context *ctx,
                               char const *restrict in, idx_t inlen,
                               char *restrict out, idx_t *outlen);
#endif

```

```

#if BASE_TYPE == 42

static int
base64_length_wrapper (int len)
{
    return BASE64_LENGTH (len);
}

static void
base64_decode_ctx_init_wrapper (struct base_decode_context *ctx)
{
    base64_decode_ctx_init (&ctx->ctx.base64);
}

static bool
base64_decode_ctx_wrapper (struct base_decode_context *ctx,
                           char const *restrict in, idx_t inlen,
                           char *restrict out, idx_t *outlen)
{
    bool b = base64_decode_ctx (&ctx->ctx.base64, in, inlen, out, outlen);
    ctx->i = ctx->ctx.base64.i;
    return b;
}

static void
init_inbuf (struct base_decode_context *ctx)
{
    ctx->bufsize = DEC_BLOCKSIZE;
    ctx->inbuf = xcharalloc (ctx->bufsize);
}

static void

```

```

prepare_inbuf (struct base_decode_context *ctx, idx_t inlen)
{
if (ctx->bufsize < inlen)
{
    ctx->bufsize = inlen * 2;
    ctx->inbuf = xnrealloc (ctx->inbuf, ctx->bufsize, sizeof (char));
}
}

static void
base64url_encode (char const *restrict in, idx_t inlen,
                  char *restrict out, idx_t outlen)
{
base64_encode (in, inlen, out, outlen);
/* translate 62nd and 63rd characters */
char *p = out;
while (outlen--)
{
    if (*p == '+')
        *p = '-';
    else if (*p == '/')
        *p = '_';
    ++p;
}
}

static bool
isubase64url (unsigned char ch)
{
return (ch == '-' || ch == '_'
        || (ch != '+' && ch != '/' && isubase64 (ch)));
}

static void
base64url_decode_ctx_init_wrapper (struct base_decode_context *ctx)
{
base64_decode_ctx_init (&ctx->ctx.base64);
init_inbuf (ctx);
}

static bool
base64url_decode_ctx_wrapper (struct base_decode_context *ctx,
                             char const *restrict in, idx_t inlen,
                             char *restrict out, idx_t *outlen)
{
prepare_inbuf (ctx, inlen);
memcpy (ctx->inbuf, in, inlen);

/* translate 62nd and 63rd characters */
idx_t i = inlen;
char *p = ctx->inbuf;
while (i--)
{
    if (*p == '+' || *p == '/')
    {
        *outlen = 0;
        return false; /* reject base64 input */
    }
}
}

```

```

    else if (*p == '-')
        *p = '+';
    else if (*p == '_')
        *p = '/';
    ++p;
}

bool b = base64_decode_ctx (&ctx->ctx.base64, ctx->inbuf, inlen,
                           out, outlen);
ctx->i = ctx->ctx.base64.i;

return b;
}

static int
base32_length_wrapper (int len)
{
return BASE32_LENGTH (len);
}

static void
base32_decode_ctx_init_wrapper (struct base_decode_context *ctx)
{
base32_decode_ctx_init (&ctx->ctx.base32);
}

static bool
base32_decode_ctx_wrapper (struct base_decode_context *ctx,
                           char const *restrict in, idx_t inlen,
                           char *restrict out, idx_t *outlen)
{
bool b = base32_decode_ctx (&ctx->ctx.base32, in, inlen, out, outlen);
ctx->i = ctx->ctx.base32.i;
return b;
}

/* ABCDEFGHIJKLMNOPQRSTUVWXYZ234567
   to
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ */
static const char base32_norm_to_hex[32 + 9] = {
/*0x32, 0x33, 0x34, 0x35, 0x36, 0x37, */
'Q', 'R', 'S', 'T', 'U', 'V',
0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f, 0x40,
/*0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, */
'0', '1', '2', '3', '4', '5', '6', '7',
/*0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f, 0x50, */
'8', '9', 'A', 'B', 'C', 'D', 'E', 'F',
/*0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, */
'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
/*0x59, 0x5a, */
'O', 'P',
};

```

```

/* 0123456789ABCDEFHIJKLMNOPQRSTUVWXYZ
   to
ABCDEFGHIJKLMNOPQRSTUVWXYZ234567 */
static const char base32_hex_to_norm[32 + 9] = {
/* from: 0x30 .. 0x39 ('0' to '9') */
/* to:*/ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f, 0x40,
/* from: 0x41 .. 0x4A ('A' to 'J') */
/* to:*/ 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
/* from: 0x4B .. 0x54 ('K' to 'T') */
/* to:*/ 'U', 'V', 'W', 'X', 'Y', 'Z', '2', '3', '4', '5',
/* from: 0x55 .. 0x56 ('U' to 'V') */
/* to:*/ '6', '7'
};

inline static bool
isbase32hex (unsigned char ch)
{
return ('0' <= ch && ch <= '9') || ('A' <= ch && ch <= 'V');
}

static void
base32hex_encode (char const *restrict in, idx_t inlen,
                  char *restrict out, idx_t outlen)
{
base32_encode (in, inlen, out, outlen);

for (char *p = out; outlen--; p++)
{
    affirm (0x32 <= *p && *p <= 0x5a); /* LCOV_EXCL_LINE */
    *p = base32_norm_to_hex[*p - 0x32];
}
}

static void
base32hex_decode_ctx_init_wrapper (struct base_decode_context *ctx)
{
base32_decode_ctx_init (&ctx->ctx.base32);
init_inbuf (ctx);
}

static bool
base32hex_decode_ctx_wrapper (struct base_decode_context *ctx,
                             char const *restrict in, idx_t inlen,
                             char *restrict out, idx_t *outlen)
{
prepare_inbuf (ctx, inlen);

idx_t i = inlen;
char *p = ctx->inbuf;
while (i--)
{

```

```

if (isubbase32hex (*in))
    *p = base32_hex_to_norm[*in - 0x30];
else
    *p = *in;
++p;
++in;
}

bool b = base32_decode_ctx (&ctx->ctx.base32, ctx->inbuf, inlen,
                           out, outlen);
ctx->i = ctx->ctx.base32.i;

return b;
}
/* With this approach this file works independent of the charset used
(think EBCDIC). However, it does assume that the characters in the
Base32 alphabet (A-Z2-7) are encoded in 0..255. POSIX
1003.1-2001 require that char and unsigned char are 8-bit
quantities, though, taking care of that problem. But this may be a
potential problem on non-POSIX C99 platforms.

IBM C V6 for AIX mishandles "#define B32(x) ...'x'...", so use "_" as the formal parameter rather than "x". */
#define B16(_)
(_ == '0' ? 0
 : (_ == '1' ? 1
 : (_ == '2' ? 2
 : (_ == '3' ? 3
 : (_ == '4' ? 4
 : (_ == '5' ? 5
 : (_ == '6' ? 6
 : (_ == '7' ? 7
 : (_ == '8' ? 8
 : (_ == '9' ? 9
 : (_ == 'A' || (_ == 'a' ? 10
 : (_ == 'B' || (_ == 'b' ? 11
 : (_ == 'C' || (_ == 'c' ? 12
 : (_ == 'D' || (_ == 'd' ? 13
 : (_ == 'E' || (_ == 'e' ? 14
 : (_ == 'F' || (_ == 'f' ? 15
 : -1)
static signed char const base16_to_int[256] = {
B16 (0), B16 (1), B16 (2), B16 (3),
B16 (4), B16 (5), B16 (6), B16 (7),
B16 (8), B16 (9), B16 (10), B16 (11),
B16 (12), B16 (13), B16 (14), B16 (15),
B16 (16), B16 (17), B16 (18), B16 (19),
B16 (20), B16 (21), B16 (22), B16 (23),
B16 (24), B16 (25), B16 (26), B16 (27),
B16 (28), B16 (29), B16 (30), B16 (31),
B16 (32), B16 (33), B16 (34), B16 (35),
B16 (36), B16 (37), B16 (38), B16 (39),
B16 (40), B16 (41), B16 (42), B16 (43),
B16 (44), B16 (45), B16 (46), B16 (47),
B16 (48), B16 (49), B16 (50), B16 (51),
B16 (52), B16 (53), B16 (54), B16 (55),
B16 (56), B16 (57), B16 (58), B16 (59),
B16 (60), B16 (61), B16 (62), B16 (63),
B16 (64), B16 (65), B16 (66), B16 (67),

```

```
B16 (68), B16 (69), B16 (70), B16 (71),
B16 (72), B16 (73), B16 (74), B16 (75),
B16 (76), B16 (77), B16 (78), B16 (79),
B16 (80), B16 (81), B16 (82), B16 (83),
B16 (84), B16 (85), B16 (86), B16 (87),
B16 (88), B16 (89), B16 (90), B16 (91),
B16 (92), B16 (93), B16 (94), B16 (95),
B16 (96), B16 (97), B16 (98), B16 (99),
B16 (100), B16 (101), B16 (102), B16 (103),
B16 (104), B16 (105), B16 (106), B16 (107),
B16 (108), B16 (109), B16 (110), B16 (111),
B16 (112), B16 (113), B16 (114), B16 (115),
B16 (116), B16 (117), B16 (118), B16 (119),
B16 (120), B16 (121), B16 (122), B16 (123),
B16 (124), B16 (125), B16 (126), B16 (127),
B16 (128), B16 (129), B16 (130), B16 (131),
B16 (132), B16 (133), B16 (134), B16 (135),
B16 (136), B16 (137), B16 (138), B16 (139),
B16 (140), B16 (141), B16 (142), B16 (143),
B16 (144), B16 (145), B16 (146), B16 (147),
B16 (148), B16 (149), B16 (150), B16 (151),
B16 (152), B16 (153), B16 (154), B16 (155),
B16 (156), B16 (157), B16 (158), B16 (159),
B16 (160), B16 (161), B16 (162), B16 (163),
B16 (164), B16 (165), B16 (166), B16 (167),
B16 (168), B16 (169), B16 (170), B16 (171),
B16 (172), B16 (173), B16 (174), B16 (175),
B16 (176), B16 (177), B16 (178), B16 (179),
B16 (180), B16 (181), B16 (182), B16 (183),
B16 (184), B16 (185), B16 (186), B16 (187),
B16 (188), B16 (189), B16 (190), B16 (191),
B16 (192), B16 (193), B16 (194), B16 (195),
B16 (196), B16 (197), B16 (198), B16 (199),
B16 (200), B16 (201), B16 (202), B16 (203),
B16 (204), B16 (205), B16 (206), B16 (207),
B16 (208), B16 (209), B16 (210), B16 (211),
B16 (212), B16 (213), B16 (214), B16 (215),
B16 (216), B16 (217), B16 (218), B16 (219),
B16 (220), B16 (221), B16 (222), B16 (223),
B16 (224), B16 (225), B16 (226), B16 (227),
B16 (228), B16 (229), B16 (230), B16 (231),
B16 (232), B16 (233), B16 (234), B16 (235),
B16 (236), B16 (237), B16 (238), B16 (239),
B16 (240), B16 (241), B16 (242), B16 (243),
B16 (244), B16 (245), B16 (246), B16 (247),
B16 (248), B16 (249), B16 (250), B16 (251),
B16 (252), B16 (253), B16 (254), B16 (255)
};
```

```
static bool
isbase16 (unsigned char ch)
{
    return ch < sizeof base16_to_int && 0 <= base16_to_int[ch];
}
```

```
static int
base16_length (int len)
{
    return len * 2;
}
```

```

static void
base16_encode (char const *restrict in, idx_t inlen,
               char *restrict out, idx_t outlen)
{
static const char base16[16] = "0123456789ABCDEF";
while (inlen && outlen)
{
    unsigned char c = *in;
    *out++ = base16[c >> 4];
    *out++ = base16[c & 0x0F];
    ++in;
    inlen--;
    outlen -= 2;
}
}

static void
base16_decode_ctx_init (struct base_decode_context *ctx)
{
init_inbuf (ctx);
ctx->ctx.base16.nibble = -1;
ctx->i = 1;
}

static bool
base16_decode_ctx (struct base_decode_context *ctx,
                  char const *restrict in, idx_t inlen,
                  char *restrict out, idx_t *outlen)
{
bool ignore_lines = true; /* for now, always ignore them */
char *out0 = out;
signed char nibble = ctx->ctx.base16.nibble;

/* inlen==0 is request to flush output.
   if there is a dangling high nibble - we are missing the low nibble,
   so return false - indicating an invalid input. */
if (inlen == 0)
{
    *
    *outlen = 0;
    return nibble < 0;
}

while (inlen--)
{
    unsigned char c = *in++;
    if (ignore_lines && c == '\n')
        continue;

    if (sizeof base16_to_int <= c || base16_to_int[c] < 0)
    {
        *
        *outlen = out - out0;
        return false; /* garbage - return false */
    }

    if (nibble < 0)

```

```

        nibble = base16_to_int[c];
    else
    {
        /* have both nibbles, write octet */
        *out++ = (nibble << 4) + base16_to_int[c];
        nibble = -1;
    }
}

ctx->ctx.base16.nibble = nibble;
*outlen = out - out0;
return true;
}

static int
z85_length (int len)
{
/* Z85 does not allow padding, so no need to round to highest integer. */
int outlen = (len * 5) / 4;
return outlen;
}

static bool
isuz85 (unsigned char ch)
{
return c_isalnum (ch) || strchr (".-:+^!/*?&<>@%$#", ch) != nullptr;
}

static char const z85_encoding[85] =
"0123456789"
"abcdefghijklmnopqrstuvwxyz"
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
".-:+^!/*?&<>@%$#";

static void
z85_encode (char const *restrict in, idx_t inlen,
            char *restrict out, idx_t outlen)
{
int i = 0;
unsigned char quad[4];
idx_t outidx = 0;

while (true)
{
if (inlen == 0)
{
/* no more input, exactly on 4 octet boundary. */
if (i == 0)
return;

/* currently, there's no way to return an error in encoding. */
error (EXIT_FAILURE, 0,
      _("invalid input (length must be multiple of 4 characters")));
}
else
{
quad[i++] = *in++;
}
}

```

```

--inlen;
}

/* Got a quad, encode it */
if (i == 4)
{
    int_fast64_t val = quad[0];
    val = (val << 24) + (quad[1] << 16) + (quad[2] << 8) + quad[3];

    for (int j = 4; j >= 0; --j)
    {
        int c = val % 85;
        val /= 85;

        /* NOTE: if there is padding (which is trimmed by z85
           before outputting the result), the output buffer 'out'
           might not include enough allocated bytes for the padding,
           so don't store them. */
        if (outidx + j < outlen)
            out[j] = z85_encoding[c];
    }
    out += 5;
    outidx += 5;
    i = 0;
}
}

static void
z85_decode_ctx_init (struct base_decode_context *ctx)
{
init_inbuf (ctx);
ctx->ctx.z85.i = 0;
ctx->i = 1;
}

#define Z85_LO_CTX_TO_32BIT_VAL(ctx) \
(((ctx)->ctx.z85.octets[1] * 85 * 85 * 85) + \
((ctx)->ctx.z85.octets[2] * 85 * 85) + \
((ctx)->ctx.z85.octets[3] * 85) + \
((ctx)->ctx.z85.octets[4]))

#define Z85_HI_CTX_TO_32BIT_VAL(ctx) \
((int_fast64_t) (ctx)->ctx.z85.octets[0] * 85 * 85 * 85 * 85 )

/*
0 - 9: 0 1 2 3 4 5 6 7 8 9
10 - 19: a b c d e f g h i j
20 - 29: k l m n o p q r s t
30 - 39: u v w x y z A B C D
40 - 49: E F G H I J K L M N
50 - 59: O P Q R S T U V W X
60 - 69: Y Z . - : + = ^ ! / #dummy comment to workaround syntax-check
70 - 79: * ? & < > () [] {
80 - 84: } @ % $ #
*/
static signed char const z85_decoding[93] = {
68, -1, 84, 83, 82, 72, -1, /* ! " # $ % & ' */

```

```

75, 76, 70, 65, -1, 63, 62, 69,      /* () * + , - . / */
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, /* '0' to '9' */
64, -1, 73, 66, 74, 71, 81,      /* : ; < = > ? @ */
36, 37, 38, 39, 40, 41, 42, 43, 44, 45, /* 'A' to 'J' */
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, /* 'K' to 'T' */
56, 57, 58, 59, 60, 61,      /* 'U' to 'Z' */
77, -1, 78, 67, -1, -1,      /* [ \ ] ^ _ ` */
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, /* 'a' to 'j' */
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, /* 'k' to 't' */
30, 31, 32, 33, 34, 35,      /* 'u' to 'z' */
79, -1, 80      /* { | } */
};

static bool
z85_decode_ctx (struct base_decode_context *ctx,
                char const *restrict in, idx_t inlen,
                char *restrict out, idx_t *outlen)
{
bool ignore_lines = true; /* for now, always ignore them */

*outlen = 0;

/* inlen==0 is request to flush output.
   if there are dangling values - we are missing entries,
   so return false - indicating an invalid input. */
if (inlen == 0)
{
if (ctx->ctx.z85.i > 0)
{
/* Z85 variant does not allow padding - input must
   be a multiple of 5 - so return error. */
return false;
}
return true;
}

while (inlen--)
{
if (ignore_lines && *in == '\n')
{
++in;
continue;
}

/* z85 decoding */
unsigned char c = *in;

if (c >= 33 && c <= 125)
{
signed char ch = z85_decoding[c - 33];
if (ch < 0)
    return false; /* garbage - return false */
c = ch;
}
else
    return false; /* garbage - return false */

++in;

ctx->ctx.z85.octets[ctx->ctx.z85.i++] = c;
}

```

```

if (ctx->ctx.z85.i == 5)
{
    /* decode the lowest 4 octets, then check for overflows. */
    int_fast64_t val = Z85_LO_CTX_TO_32BIT_VAL (ctx);

    /* The Z85 spec and the reference implementation say nothing
       about overflows. To be on the safe side, reject them. */

    val += Z85_HI_CTX_TO_32BIT_VAL (ctx);
    if ((val >> 24) & ~0xFF)
        return false;

    *out++ = val >> 24;
    *out++ = (val >> 16) & 0xFF;
    *out++ = (val >> 8) & 0xFF;
    *out++ = val & 0xFF;

    *outlen += 4;

    ctx->ctx.z85.i = 0;
}
}

ctx->i = ctx->ctx.z85.i;
return true;
}

inline static bool
isubase2 (unsigned char ch)
{
return ch == '0' || ch == '1';
}

static int
base2_length (int len)
{
return len * 8;
}

inline static void
base2msbf_encode (char const *restrict in, idx_t inlen,
                  char *restrict out, idx_t outlen)
{
while (inlen && outlen)
{
    unsigned char c = *in;
    for (int i = 0; i < 8; i++)
    {
        *out++ = c & 0x80 ? '1' : '0';
        c <<= 1;
    }
    inlen--;
    outlen -= 8;
    ++in;
}
}

inline static void
base2lsbf_encode (char const *restrict in, idx_t inlen,

```

```

        char *restrict out, idx_t outlen)
{
while (inlen && outlen)
{
    unsigned char c = *in;
    for (int i = 0; i < 8; i++)
    {
        *out++ = c & 0x01 ? '1' : '0';
        c >>= 1;
    }
    inlen--;
    outlen -= 8;
    ++in;
}
}

static void
base2_decode_ctx_init (struct base_decode_context *ctx)
{
init_inbuf (ctx);
ctx->ctx.base2.octet = 0;
ctx->i = 0;
}

static bool
base2lsbf_decode_ctx (struct base_decode_context *ctx,
                      char const *restrict in, idx_t inlen,
                      char *restrict out, idx_t *outlen)
{
bool ignore_lines = true; /* for now, always ignore them */

*outlen = 0;

/* inlen==0 is request to flush output.
   if there is a dangling bit - we are missing some bits,
   so return false - indicating an invalid input. */
if (inlen == 0)
    return ctx->i == 0;

while (inlen--)
{
    if (ignore_lines && *in == '\n')
    {
        ++in;
        continue;
    }

    if (!isbase2 (*in))
        return false;

    bool bit = (*in == '1');
    ctx->ctx.base2.octet |= bit << ctx->i;
    ++ctx->i;

    if (ctx->i == 8)
    {
        *out++ = ctx->ctx.base2.octet;
        ctx->ctx.base2.octet = 0;
    }
}
}

```

```

++*outlen;
ctx->i = 0;
}

++in;
}

return true;
}

static bool
base2msbf_decode_ctx (struct base_decode_context *ctx,
                      char const *restrict in, idx_t inlen,
                      char *restrict out, idx_t *outlen)
{
bool ignore_lines = true; /* for now, always ignore them */

*outlen = 0;

/* inlen==0 is request to flush output.
   if there is a dangling bit - we are missing some bits,
   so return false - indicating an invalid input. */
if (inlen == 0)
    return ctx->i == 0;

while (inlen--)
{
    if (ignore_lines && *in == '\n')
    {
        ++in;
        continue;
    }

    if (!isubbase2 (*in))
        return false;

    bool bit = (*in == '1');
    if (ctx->i == 0)
        ctx->i = 8;
    --ctx->i;
    ctx->ctx.base2.octet |= bit << ctx->i;

    if (ctx->i == 0)
    {
        *out++ = ctx->ctx.base2.octet;
        ctx->ctx.base2.octet = 0;
        ++*outlen;
        ctx->i = 0;
    }

    ++in;
}

return true;
}

#endif /* BASE_TYPE == 42, i.e., "basenc"*/

```

```

static void
wrap_write (char const *buffer, idx_t len,
            idx_t wrap_column, idx_t *current_column, FILE *out)
{
if (wrap_column == 0)
{
/* Simple write. */
if (fwrite (buffer, 1, len, stdout) < len)
    write_error ();
}
else
for (idx_t written = 0; written < len; )
{
    idx_t to_write = MIN (wrap_column - *current_column, len - written);

    if (to_write == 0)
    {
        if (fputc ('\n', out) == EOF)
        write_error ();
        *current_column = 0;
    }
    else
    {
        if (fwrite (buffer + written, 1, to_write, stdout) < to_write)
        write_error ();
        *current_column += to_write;
        written += to_write;
    }
}
}

static _Noreturn void
finish_and_exit (FILE *in, char const *infile)
{
if (fclose (in) != 0)
{
    if (STREQ (infile, "-"))
        error (EXIT_FAILURE, errno, _("closing standard input"));
    else
        error (EXIT_FAILURE, errno, "%s", quotef (infile));
}

exit (EXIT_SUCCESS);
}

static _Noreturn void
do_encode (FILE *in, char const *infile, FILE *out, idx_t wrap_column)
{
idx_t current_column = 0;
char *inbuf, *outbuf;
idx_t sum;

inbuf = xmalloc (ENC_BLOCKSIZE);
outbuf = xmalloc (BASE_LENGTH (ENC_BLOCKSIZE));

do
{
    idx_t n;

    sum = 0;

```

```

do
{
    n = fread (inbuf + sum, 1, ENC_BLOCKSIZE - sum, in);
    sum += n;
}
while (!feof (in) && !ferror (in) && sum < ENC_BLOCKSIZE);

if (sum > 0)
{
    /* Process input one block at a time. Note that ENC_BLOCKSIZE
       is sized so that no pad chars will appear in output.*/
    base_encode (inbuf, sum, outbuf, BASE_LENGTH (sum));

    wrap_write (outbuf, BASE_LENGTH (sum), wrap_column,
                &current_column, out);
}
}

while (!feof (in) && !ferror (in) && sum == ENC_BLOCKSIZE);

/* When wrapping, terminate last line.*/
if (wrap_column && current_column > 0 && fputc ('\n', out) == EOF)
    write_error ();

if (ferror (in))
    error (EXIT_FAILURE, errno, _("read error"));

finish_and_exit (in, infile);
}

static _Noreturn void
do_decode (FILE *in, char const *infile, FILE *out, bool ignore_garbage)
{
char *inbuf, *outbuf;
idx_t sum;
struct base_decode_context ctx;

char padbuf[8] = "=====";
inbuf = xmalloc (BASE_LENGTH (DEC_BLOCKSIZE));
outbuf = xmalloc (DEC_BLOCKSIZE);

#if BASE_TYPE == 42
ctx.inbuf = nullptr;
#endif
base_decode_ctx_init (&ctx);

do
{
    bool ok;

    sum = 0;
    do
    {
        idx_t n = fread (inbuf + sum,
                        1, BASE_LENGTH (DEC_BLOCKSIZE) - sum, in);

        if (ignore_garbage)
        {
            for (idx_t i = 0; n > 0 && i < n;)
            {
                if (isubase (inbuf[sum + i]) || inbuf[sum + i] == '=')

```

```

        i++;
    else
        memmove (inbuf + sum + i, inbuf + sum + i + 1, --n - i);
    }
}

sum += n;

if (ferror (in))
    error (EXIT_FAILURE, errno, _("read error"));
}
while (sum < BASE_LENGTH (DEC_BLOCKSIZE) && !feof (in));

/* The following "loop" is usually iterated just once.
   However, when it processes the final input buffer, we want
   to iterate it one additional time, but with an indicator
   telling it to flush what is in CTX. */
for (int k = 0; k < 1 + !!feof (in); k++)
{
    if (k == 1)
    {
        if (ctx.i == 0)
            break;

        /* auto pad input (at eof). */
        idx_t auto_padding = REQUIRED_PADDING (ctx.i);
        if (auto_padding && (sum == 0 || inbuf[sum - 1] != '='))
        {
            affirm (auto_padding <= sizeof (padbuf));
            IF_LINT (free (inbuf));
            sum = auto_padding;
            inbuf = padbuf;
        }
        else
            sum = 0; /* process ctx buffer only */
    }
    idx_t n = DEC_BLOCKSIZE;
    ok = base_decode_ctx (&ctx, inbuf, sum, outbuf, &n);

    if (fwrite (outbuf, 1, n, out) < n)
        write_error ();

    if (!ok)
        error (EXIT_FAILURE, 0, _("invalid input"));
    }
}
while (!feof (in));

finish_and_exit (in, infile);
}

int
main (int argc, char **argv)
{
int opt;
FILE *input_fh;
char const *infile;

/* True if --decode has been given and we should decode data. */
bool decode = false;

```

```

/* True if we should ignore non-base-alphabetic characters. */
bool ignore_garbage = false;
/* Wrap encoded data around the 76th column, by default. */
idx_t wrap_column = 76;

#if BASE_TYPE == 42
int base_type = 0;
#endif

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

atexit (close_stdout);

while ((opt = getopt_long (argc, argv, "diw:", long_options, nullptr)) != -1)
    switch (opt)
    {
    case 'd':
        decode = true;
        break;

    case 'w':
    {
        intmax_t w;
        strtol_error s_err = xstrtoimax (optarg, nullptr, 10, &w, "");
        if (LONGINT_OVERFLOW < s_err || w < 0)
            error (EXIT_FAILURE, 0, "%s: %s",
                   _("invalid wrap size"), quote (optarg));
        wrap_column = s_err == LONGINT_OVERFLOW || IDX_MAX < w ? 0 : w;
    }
        break;

    case 'i':
        ignore_garbage = true;
        break;

#if BASE_TYPE == 42
        case BASE64_OPTION:
        case BASE64URL_OPTION:
        case BASE32_OPTION:
        case BASE32HEX_OPTION:
        case BASE16_OPTION:
        case BASE2MSBF_OPTION:
        case BASE2LSBF_OPTION:
        case Z85_OPTION:
            base_type = opt;
            break;
#endif

        case_GETOPT_HELP_CHAR;

        case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

        default:
            usage (EXIT_FAILURE);
            break;
    }

```

```

#if BASE_TYPE == 42
switch (base_type)
{
    case BASE64_OPTION:
        base_length = base64_length_wrapper;
        required_padding = base64_required_padding;
        isubase = isubase64;
        base_encode = base64_encode;
        base_decode_ctx_init = base64_decode_ctx_init_wrapper;
        base_decode_ctx = base64_decode_ctx_wrapper;
        break;

    case BASE64URL_OPTION:
        base_length = base64_length_wrapper;
        required_padding = base64_required_padding;
        isubase = isubase64url;
        base_encode = base64url_encode;
        base_decode_ctx_init = base64url_decode_ctx_init_wrapper;
        base_decode_ctx = base64url_decode_ctx_wrapper;
        break;

    case BASE32_OPTION:
        base_length = base32_length_wrapper;
        required_padding = base32_required_padding;
        isubase = isubase32;
        base_encode = base32_encode;
        base_decode_ctx_init = base32_decode_ctx_init_wrapper;
        base_decode_ctx = base32_decode_ctx_wrapper;
        break;

    case BASE32HEX_OPTION:
        base_length = base32_length_wrapper;
        required_padding = base32_required_padding;
        isubase = isubase32hex;
        base_encode = base32hex_encode;
        base_decode_ctx_init = base32hex_decode_ctx_init_wrapper;
        base_decode_ctx = base32hex_decode_ctx_wrapper;
        break;

    case BASE16_OPTION:
        base_length = base16_length;
        required_padding = no_required_padding;
        isubase = isubase16;
        base_encode = base16_encode;
        base_decode_ctx_init = base16_decode_ctx_init;
        base_decode_ctx = base16_decode_ctx;
        break;

    case BASE2MSBF_OPTION:
        base_length = base2_length;
        required_padding = no_required_padding;
        isubase = isubase2;
        base_encode = base2msbf_encode;
        base_decode_ctx_init = base2_decode_ctx_init;
        base_decode_ctx = base2msbf_decode_ctx;
        break;

    case BASE2LSBF_OPTION:
        base_length = base2_length;

```

```

required_padding = no_required_padding;
isubase = isubase2;
base_encode = base2lbf_encode;
base_decode_ctx_init = base2_decode_ctx_init;
base_decode_ctx = base2lbf_decode_ctx;
break;

case Z85_OPTION:
base_length = z85_length;
required_padding = no_required_padding;
isubase = isuz85;
base_encode = z85_encode;
base_decode_ctx_init = z85_decode_ctx_init;
base_decode_ctx = z85_decode_ctx;
break;

default:
error (0, 0, _("missing encoding type"));
usage (EXIT_FAILURE);
}
#endif

if (argc - optind > 1)
{
error (0, 0, _("extra operand %s"), quote (argv[optind + 1]));
usage (EXIT_FAILURE);
}

if (optind < argc)
    infile = argv[optind];
else
    infile = "-";

if (STREQ (infile, "-"))
{
xset_binary_mode (STDIN_FILENO, O_BINARY);
input_fh = stdin;
}
else
{
input_fh = fopen (infile, "rb");
if (input_fh == nullptr)
    error (EXIT_FAILURE, errno, "%s", quotef (infile));
}
#ifndef __MVS__
__disableautocvt(fileno(input_fh));
#endif
fadvise (input_fh, FADVISE_SEQUENTIAL);

if (decode)
    do_decode (input_fh, infile, stdout, ignore_garbage);
else
    do_encode (input_fh, infile, stdout, wrap_column);
}

"""

"patch":
"""

diff --git a/src/basenc.c b/src/basenc.c
index ce259c4..32378fc 100644
--- a/src/basenc.c

```

```

+++ b/src/basenc.c
@@ -1235,7 +1235,9 @@ main (int argc, char **argv)
    if (input_fh == nullptr)
        error (EXIT_FAILURE, errno, "%s", quotef (infile));
    }
-
+#ifdef __MVS__
+ _disableautocvt(fileno(input_fh));
+#endif
fadvise (input_fh, FADVISE_SEQUENTIAL);

    if (decode)
"""
},
{
    "wrong_code": """
/* cat -- concatenate files and print on the standard output.
Copyright (C) 1988-2024 Free Software Foundation, Inc.
```

This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License  
along with this program. If not, see <<https://www.gnu.org/licenses/>>. \*/

/\* Differences from the Unix cat:  
\* Always unbuffered, -u is ignored.  
\* Usually much faster than other versions of cat, the difference  
is especially apparent when using the -v option.

By tege@sics.se, Torbjörn Granlund, advised by rms, Richard Stallman. \*/

```
#include <config.h>

#include <stdio.h>
#include <getopt.h>
#include <sys/types.h>

#if HAVE_STROPTS_H
# include <stropts.h>
#endif
#include <sys/ioctl.h>

#include "system.h"
#include "alignalloc.h"
#include "ioblksize.h"
#include "fadvise.h"
#include "full-write.h"
#include "safe-read.h"
#include "xbinary-io.h"
```

```

/* The official name of this program (e.g., no 'g' prefix). */
#define PROGRAM_NAME "cat"

#define AUTHORS \
proper_name_lite ("Torbjorn Granlund", "Torbj\303\266rn Granlund"), \
proper_name ("Richard M. Stallman")

/* Name of input file. May be "-". */
static char const *infile;

/* Descriptor on which input file is open. */
static int input_desc;

/* Buffer for line numbers.
An 11 digit counter may overflow within an hour on a P2/466,
an 18 digit counter needs about 1000y */
#define LINE_COUNTER_BUF_LEN 20
static char line_buf[LINE_COUNTER_BUF_LEN] =
{
    ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
    ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '0',
    '\t', '\0'
};

/* Position in 'line_buf' where printing starts. This will not change
unless the number of lines is larger than 999999. */
static char *line_num_print = line_buf + LINE_COUNTER_BUF_LEN - 8;

/* Position of the first digit in 'line_buf'. */
static char *line_num_start = line_buf + LINE_COUNTER_BUF_LEN - 3;

/* Position of the last digit in 'line_buf'. */
static char *line_num_end = line_buf + LINE_COUNTER_BUF_LEN - 3;

/* Preserves the 'cat' function's local 'newlines' between invocations. */
static int newlines2 = 0;

/* Whether there is a pending CR to process. */
static bool pending_cr = false;

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    {
        printf (_("\
Usage: %s [OPTION]... [FILE]...\n\
")),
        program_name);
    fputs (_("\
Concatenate FILE(s) to standard output.\n\
"), stdout);

    emit_stdin_note ();

    fputs (_("\
\n\
-A, --show-all      equivalent to -vET\n\

```

```

-b, --number-nonblank  number nonempty output lines, overrides -n\n\
-e                  equivalent to -vE\n\
-E, --show-ends      display $ at end of each line\n\
-n, --number         number all output lines\n\
-s, --squeeze-blank suppress repeated empty output lines\n\
"), stdout);
    fputs (_("\
-t                  equivalent to -vT\n\
-T, --show-tabs     display TAB characters as ^I\n\
-u                  (ignored)\n\
-v, --show-nonprinting use ^ and M- notation, except for LFD and TAB\n\
"), stdout);
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
    printf (_("\
\n\
Examples:\n\
%> f - g  Output f's contents, then standard input, then g's contents.\n\
%> s      Copy standard input to standard output.\n\
"),
            program_name, program_name);
    emit_ancillary_info (PROGRAM_NAME);
}
exit (status);
}

/* Compute the next line number. */

static void
next_line_num (void)
{
char *endp = line_num_end;
do
{
    if ((*endp)++ < '9')
        return;
    *endp-- = '0';
}
while (endp >= line_num_start);

if (line_num_start > line_buf)
    *--line_num_start = '1';
else
    *line_buf = '>';
if (line_num_start < line_num_print)
    line_num_print--;
}

/* Plain cat. Copy the file behind 'input_desc' to STDOUT_FILENO.
BUF (of size BUFSIZE) is the I/O buffer, used by reads and writes.
Return true if successful. */

static bool
simple_cat (char *buf, idx_t bufsize)
{
/* Loop until the end of the file. */

while (true)
{
/* Read a block of input. */

```

```

size_t n_read = safe_read (input_desc, buf, bufsize);
if (n_read == SAFE_READ_ERROR)
{
    error (0, errno, "%s", quotef (infile));
    return false;
}

/* End of this file? */

if (n_read == 0)
    return true;

/* Write this block out. */

if (full_write (STDOUT_FILENO, buf, n_read) != n_read)
    write_error ();
}

/* Write any pending output to STDOUT_FILENO.
Pending is defined to be the *BPOUT - OUTBUF bytes starting at OUTBUF.
Then set *BPOUT to OUTPUT if it's not already that value. */

static inline void
write_pending (char *outbuf, char **bpout)
{
idx_t n_write = *bpout - outbuf;
if (0 < n_write)
{
    if (full_write (STDOUT_FILENO, outbuf, n_write) != n_write)
        write_error ();
    *bpout = outbuf;
}
}

/* Copy the file behind 'input_desc' to STDOUT_FILENO.
Use INBUF and read INSIZE with each call,
and OUTBUF and write OUTSIZE with each call.
(The buffers are a bit larger than the I/O sizes.)
The remaining boolean args say what 'cat' options to use.

```

Return true if successful.  
Called if any option more than -u was specified.

A newline character is always put at the end of the buffer, to make  
an explicit test for buffer end unnecessary. \*/

```

static bool
cat (char *inbuf, idx_t insize, char *outbuf, idx_t outsize,
    bool show_nonprinting, bool show_tabs, bool number, bool number_nonblank,
    bool show_ends, bool squeeze_blank)
{
/* Last character read from the input buffer. */
unsigned char ch;

/* Determines how many consecutive newlines there have been in the
input. 0 newlines makes NEWLINES -1, 1 newline makes NEWLINES 1,
etc. Initially 0 to indicate that we are at the beginning of a
new line. The "state" of the procedure is determined by

```

```

    NEWLINES. */
int newlines = newlines2;

#ifndef FIONREAD
/* If nonzero, use the FIONREAD ioctl, as an optimization.
   (On Ultrix, it is not supported on NFS file systems.) */
bool use_fionread = true;
#endif

/* The inbuf pointers are initialized so that BPIN > EOB, and thereby input
   is read immediately. */

/* Pointer to the first non-valid byte in the input buffer, i.e., the
   current end of the buffer. */
char *eob = inbuf;

/* Pointer to the next character in the input buffer. */
char *bpin = eob + 1;

/* Pointer to the position where the next character shall be written. */
char *bpout = outbuf;

while (true)
{
do
{
/* Write if there are at least OUTSIZE bytes in OUTBUF. */

if (outbuf + outsize <= bpout)
{
char *wp = outbuf;
idx_t remaining_bytes;
do
{
if (full_write (STDOUT_FILENO, wp, outsize) != outsize)
    write_error ();
wp += outsize;
remaining_bytes = bpout - wp;
}
while (outsize <= remaining_bytes);

/* Move the remaining bytes to the beginning of the
   buffer. */

memmove (outbuf, wp, remaining_bytes);
bpout = outbuf + remaining_bytes;
}

/* Is INBUF empty? */

if (bpin > eob)
{
bool input_pending = false;
#endif FIONREAD
int n_to_read = 0;

/* Is there any input to read immediately?
   If not, we are about to wait,
   so write all buffered output before waiting. */

```

```

if (use_fionread
    && ioctl (input_desc, FIONREAD, &n_to_read) < 0)
{
/* Ultrix returns EOPNOTSUPP on NFS;
   HP-UX returns ENOTTY on pipes.
   SunOS returns EINVAL and
   More/BSD returns ENODEV on special files
   like /dev/null.
   Irix-5 returns ENOSYS on pipes. */
if (errno == EOPNOTSUPP || errno == ENOTTY
    || errno == EINVAL || errno == ENODEV
    || errno == ENOSYS)
    use_fionread = false;
else
{
    {
        error (0, errno, _("cannot do ioctl on %s"),
               quoteaf (infile));
        newlines2 = newlines;
        return false;
    }
}
if (n_to_read != 0)
    input_pending = true;
#endif

if (!input_pending)
    write_pending (outbuf, &bpout);

/* Read more input into INBUF. */

size_t n_read = safe_read (input_desc, inbuf, insize);
if (n_read == SAFE_READ_ERROR)
{
    {
        error (0, errno, "%s", quotef (infile));
        write_pending (outbuf, &bpout);
        newlines2 = newlines;
        return false;
    }
}
if (n_read == 0)
{
    {
        write_pending (outbuf, &bpout);
        newlines2 = newlines;
        return true;
    }
}

/* Update the pointers and insert a sentinel at the buffer
end. */

bpin = inbuf;
eob = bpin + n_read;
*eob = '\n';
}
else
{
    /* It was a real (not a sentinel) newline. */

    /* Was the last line empty?
       (i.e., have two or more consecutive newlines been read?) */

    if (++newlines > 0)

```

```

{
if (newlines >= 2)
{
/* Limit this to 2 here. Otherwise, with lots of
consecutive newlines, the counter could wrap
around at INT_MAX. */
newlines = 2;

/* Are multiple adjacent empty lines to be substituted
by single ditto (-s), and this was the second empty
line? */
if (squeeze_blank)
{
ch = *bpin++;
continue;
}
}

/* Are line numbers to be written at empty lines (-n)? */

if (number && !number_nonblank)
{
next_line_num ();
bpout = stpcpy (bpout, line_num_print);
}
}

/* Output a currency symbol if requested (-e). */
if (show_ends)
{
if (pending_cr)
{
*bpout++ = '^';
*bpout++ = 'M';
pending_cr = false;
}
*bpout++ = '$';
}

/* Output the newline. */

*bpout++ = '\n';
}
ch = *bpin++;
}
while (ch == '\n');

/* Here CH cannot contain a newline character. */

if (pending_cr)
{
*bpout++ = '\r';
pending_cr = false;
}

/* Are we at the beginning of a line, and line numbers are requested? */

if (newlines >= 0 && number)
{
next_line_num ();
}

```

```

bpout = stpcpy (bpout, line_num_print);
}

/* The loops below continue until a newline character is found,
   which means that the buffer is empty or that a proper newline
   has been found. */

/* If quoting, i.e., at least one of -v, -e, or -t specified,
   scan for chars that need conversion. */
if (show_nonprinting)
{
    while (true)
    {
        if (ch >= 32)
        {
            if (ch < 127)
                *bpout++ = ch;
            else if (ch == 127)
            {
                *bpout++ = '^';
                *bpout++ = '?';
            }
            else
            {
                *bpout++ = 'M';
                *bpout++ = '-';
                if (ch >= 128 + 32)
                {
                    if (ch < 128 + 127)
                        *bpout++ = ch - 128;
                    else
                    {
                        *bpout++ = '^';
                        *bpout++ = '?';
                    }
                }
                else
                {
                    *bpout++ = '^';
                    *bpout++ = ch - 128 + 64;
                }
            }
        }
        else if (ch == '\t' && !show_tabs)
            *bpout++ = '\t';
        else if (ch == '\n')
        {
            newlines = -1;
            break;
        }
        else
        {
            *bpout++ = '^';
            *bpout++ = ch + 64;
        }
        ch = *bpin++;
    }
}
else

```

```

{
/* Not quoting, neither of -v, -e, or -t specified. */
while (true)
{
    if (ch == '\t' && show_tabs)
    {
        *bpout++ = '^';
        *bpout++ = ch + 64;
    }
    else if (ch != '\n')
    {
        if (ch == '\r' && *bpin == '\n' && show_ends)
        {
            if (bpin == eob)
                pending_cr = true;
            else
            {
                *bpout++ = '^';
                *bpout++ = 'M';
            }
        }
        else
            *bpout++ = ch;
    }
    else
    {
        newlines = -1;
        break;
    }
}

ch = *bpin++;
}
}
}
}

/* Copy data from input to output using copy_file_range if possible.
Return 1 if successful, 0 if ordinary read+write should be tried,
-1 if a serious problem has been diagnosed. */

static int
copy_cat (void)
{
/* Copy at most COPY_MAX bytes at a time; this is min
   (SSIZE_MAX, SIZE_MAX) truncated to a value that is
   surely aligned well. */
ssize_t copy_max = MIN (SSIZE_MAX, SIZE_MAX) >> 30 << 30;

/* copy_file_range does not support some cases, and it
   incorrectly returns 0 when reading from the proc file
   system on the Linux kernel through at least 5.6.19 (2020),
   so fall back on read+write if the copy_file_range is
   unsupported or the input file seems empty. */

for (bool some_copied = false; ; some_copied = true)
    switch (copy_file_range (input_desc, nullptr, STDOUT_FILENO, nullptr,
                           copy_max, 0))
    {
    case 0:
        return some_copied;
    }
}

```

```

case -1:
    if (errno == ENOSYS || is_ENOTSUP (errno) || errno == EINVAL
        || errno == EBADF || errno == EXDEV || errno == ETXTBSY
        || errno == EPERM)
        return 0;
    error (0, errno, "%s", quotef (infile));
    return -1;
}
}

int
main (int argc, char **argv)
{
/* Nonzero if we have ever read standard input. */
bool have_read_stdin = false;

struct stat stat_buf;

/* Variables that are set according to the specified options. */
bool number = false;
bool number_nonblank = false;
bool squeeze_blank = false;
bool show_ends = false;
bool show_nonprinting = false;
bool show_tabs = false;
int file_open_mode = O_RDONLY;

static struct option const long_options[] =
{
    {"number-nonblank", no_argument, nullptr, 'b'},
    {"number", no_argument, nullptr, 'n'},
    {"squeeze-blank", no_argument, nullptr, 's'},
    {"show-nonprinting", no_argument, nullptr, 'v'},
    {"show-ends", no_argument, nullptr, 'E'},
    {"show-tabs", no_argument, nullptr, 'T'},
    {"show-all", no_argument, nullptr, 'A'},
    {GETOPT_HELP_OPTION_DECL},
    {GETOPT_VERSION_OPTION_DECL},
    {nullptr, 0, nullptr, 0}
};

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

/* Arrange to close stdout if we exit via the
   case_GETOPT_HELP_CHAR or case_GETOPT_VERSION_CHAR code.
   Normally STDOUT_FILENO is used rather than stdout, so
   close_stdout does nothing. */
atexit (close_stdout);

/* Parse command line options. */

int c;
while ((c = getopt_long (argc, argv, "benstuvAET", long_options, nullptr))
!= -1)

```

```

{
switch (c)
{
case 'b':
number = true;
number_nonblank = true;
break;

case 'e':
show_ends = true;
show_nonprinting = true;
break;

case 'n':
number = true;
break;

case 's':
squeeze_blank = true;
break;

case 't':
show_tabs = true;
show_nonprinting = true;
break;

case 'u':
/* We provide the -u feature unconditionally. */
break;

case 'v':
show_nonprinting = true;
break;

case 'A':
show_nonprinting = true;
show_ends = true;
show_tabs = true;
break;

case 'E':
show_ends = true;
break;

case 'T':
show_tabs = true;
break;

case_GETOPT_HELP_CHAR;

case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

default:
usage (EXIT_FAILURE);
}
}

/* Get device, i-node number, and optimal blocksize of output. */

if (fstat (STDOUT_FILENO, &stat_buf) < 0)

```

```

error (EXIT_FAILURE, errno, _("standard output"));

/* Optimal size of i/o operations of output. */
idx_t outsize = io_blksize (&stat_buf);

/* Device and I-node number of the output. */
dev_t out_dev = stat_buf.st_dev;
ino_t out_ino = stat_buf.st_ino;

/* True if the output is a regular file. */
bool out_isreg = S_ISREG (stat_buf.st_mode) != 0;

if (! (number || show_ends || squeeze_blank))
{
    file_open_mode |= O_BINARY;
    xset_binary_mode (STDOUT_FILENO, O_BINARY);
}

/* Main loop. */

infile = "-";
int argind = optind;
bool ok = true;
idx_t page_size = getpagesize ();

do
{
    if (argind < argc)
        infile = argv[argind];

    bool reading_stdin = STREQ (infile, "-");
    if (reading_stdin)
    {
        have_read_stdin = true;
        input_desc = STDIN_FILENO;
        if (file_open_mode & O_BINARY)
            xset_binary_mode (STDIN_FILENO, O_BINARY);
    }
    else
    {
        input_desc = open (infile, file_open_mode);
        if (input_desc < 0)
        {
            error (0, errno, "%s", quotef (infile));
            ok = false;
            continue;
        }
    }

    if (fstat (input_desc, &stat_buf) < 0)
    {
        error (0, errno, "%s", quotef (infile));
        ok = false;
        goto contin;
    }

    /* Optimal size of i/o operations of input. */
    idx_t insize = io_blksize (&stat_buf);

    fdadvise (input_desc, 0, 0, FADVISE_SEQUENTIAL);
}

```

```

/* Don't copy a nonempty regular file to itself, as that would
merely exhaust the output device. It's better to catch this
error earlier rather than later. */

if (out_isreg
    && stat_buf.st_dev == out_dev && stat_buf.st_ino == out_ino
    && lseek (input_desc, 0, SEEK_CUR) < stat_buf.st_size)
{
    error (0, 0, ("%s: input file is output file"), quotef (infile));
    ok = false;
    goto contin;
}

/* Pointer to the input buffer. */
char *inbuf;

/* Select which version of 'cat' to use. If any format-oriented
options were given use 'cat'; if not, use 'copy_cat' if it
works, 'simple_cat' otherwise. */

if (! (number || show_ends || show_nonprinting
       || show_tabs || squeeze_blank))
{
    int copy_cat_status =
        out_isreg && S_ISREG (stat_buf.st_mode) ? copy_cat () : 0;
    if (copy_cat_status != 0)
    {
        inbuf = nullptr;
        ok &= 0 < copy_cat_status;
    }
    else
    {
        insize = MAX (insize, outsize);
        inbuf = xalignalloc (page_size, insize);
        ok &= simple_cat (inbuf, insize);
    }
}
else
{
    /* Allocate, with an extra byte for a newline sentinel. */
    inbuf = xalignalloc (page_size, insize + 1);

    /* Why are
       (OUTSIZE - 1 + INSIZE * 4 + LINE_COUNTER_BUF_LEN)
       bytes allocated for the output buffer?

A test whether output needs to be written is done when the input
buffer empties or when a newline appears in the input. After
output is written, at most (OUTSIZE - 1) bytes will remain in the
buffer. Now INSIZE bytes of input is read. Each input character
may grow by a factor of 4 (by the prepending of M-^). If all
characters do, and no newlines appear in this block of input, we
will have at most (OUTSIZE - 1 + INSIZE * 4) bytes in the buffer.
If the last character in the preceding block of input was a
newline, a line number may be written (according to the given
options) as the first thing in the output buffer. (Done after the
new input is read, but before processing of the input begins.)
A line number requires seldom more than LINE_COUNTER_BUF_LEN
positions.
```

Align the output buffer to a page size boundary, for efficiency  
on some paging implementations. \*/

```
idx_t bufsize;
if (ckd_mul (&bufsize, insize, 4)
    || ckd_add (&bufsize, bufsize, outsize)
    || ckd_add (&bufsize, bufsize, LINE_COUNTER_BUF_LEN - 1))
    xalloc_die ();
char *outbuf = xalignalloc (page_size, bufsize);

ok &= cat (inbuf, insize, outbuf, outsize, show_nonprinting,
           show_tabs, number, number_nonblank, show_ends,
           squeeze_blank);

alignfree (outbuf);
}

alignfree (inbuf);

contin:
if (!reading_stdin && close (input_desc) < 0)
{
    error (0, errno, "%s", quotef (infile));
    ok = false;
}
while (++argind < argc);

if (pending_cr)
{
    if (full_write (STDOUT_FILENO, "\r", 1) != 1)
        write_error ();
}

if (have_read_stdin && close (STDIN_FILENO) < 0)
    error (EXIT_FAILURE, errno, _("closing standard input"));

return ok ? EXIT_SUCCESS : EXIT_FAILURE;
}

""",
"error_category": "z/OS-Specific Runtime Error",
"error": "EDC5115I Cannot locate program /bin/cat",
"correct_code": """,

/* cat -- concatenate files and print on the standard output.
Copyright (C) 1988-2024 Free Software Foundation, Inc.
```

This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License  
along with this program. If not, see <<https://www.gnu.org/licenses/>>. \*/

```

/* Differences from the Unix cat:
 * Always unbuffered, -u is ignored.
 * Usually much faster than other versions of cat, the difference
 is especially apparent when using the -v option.

By tege@sics.se, Torbjörn Granlund, advised by rms, Richard Stallman. */

#include <config.h>

#include <stdio.h>
#include <getopt.h>
#include <sys/types.h>

#if HAVE_STROPTS_H
# include <stropts.h>
#endif
#include <sys/ioctl.h>

#include "system.h"
#include "alignalloc.h"
#include "ioblksize.h"
#include "fadvise.h"
#include "full-write.h"
#include "safe-read.h"
#include "xbinary-io.h"

/* The official name of this program (e.g., no 'g' prefix). */
#define PROGRAM_NAME "cat"

#define AUTHORS \
proper_name_lite ("Torbjorn Granlund", "Torbj\303\266rn Granlund"), \
proper_name ("Richard M. Stallman")

/* Name of input file. May be "-". */
static char const *infile;

/* Descriptor on which input file is open. */
static int input_desc;

/* Buffer for line numbers.
An 11 digit counter may overflow within an hour on a P2/466,
an 18 digit counter needs about 1000y */
#define LINE_COUNTER_BUF_LEN 20
static char line_buf[LINE_COUNTER_BUF_LEN] =
{
    ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
    ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '0',
    '\t', '\0'
};

/* Position in 'line_buf' where printing starts. This will not change
unless the number of lines is larger than 999999. */
static char *line_num_print = line_buf + LINE_COUNTER_BUF_LEN - 8;

/* Position of the first digit in 'line_buf'. */
static char *line_num_start = line_buf + LINE_COUNTER_BUF_LEN - 3;

/* Position of the last digit in 'line_buf'. */
static char *line_num_end = line_buf + LINE_COUNTER_BUF_LEN - 3;

```

```

/* Preserves the 'cat' function's local 'newlines' between invocations. */
static int newlines2 = 0;

/* Whether there is a pending CR to process. */
static bool pending_cr = false;

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    {
        printf (_("\
Usage: %s [OPTION]... [FILE]...\\n\
"),
                program_name);
        fputs (_("\
Concatenate FILE(s) to standard output.\\n\
"),
                stdout);

    emit_stdin_note ();

    fputs (_("\
\\n\
-A, --show-all      equivalent to -vET\\n\
-b, --number-nonblank  number nonempty output lines, overrides -n\\n\
-e                  equivalent to -vE\\n\
-E, --show-ends      display $ at end of each line\\n\
-n, --number          number all output lines\\n\
-s, --squeeze-blank   suppress repeated empty output lines\\n\
"),
                stdout);
    fputs (_("\
-t                  equivalent to -vT\\n\
-T, --show-tabs      display TAB characters as ^\\n\
-u                  (ignored)\\n\
-v, --show-nonprinting  use ^ and M- notation, except for LFD and TAB\\n\
"),
                stdout);
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
    printf (_("\
\\n\
Examples:\\n\
%s f - g  Output f's contents, then standard input, then g's contents.\\n\
%s      Copy standard input to standard output.\\n\
"),
            program_name, program_name);
    emit_ancillary_info (PROGRAM_NAME);
}
exit (status);
}

/* Compute the next line number. */

static void
next_line_num (void)
{
char *endp = line_num_end;
do

```

```

{
    if ((*endp)++ < '9')
        return;
    *endp-- = '0';
}
while (endp >= line_num_start);

if (line_num_start > line_buf)
    *--line_num_start = '1';
else
    *line_buf = '>';
if (line_num_start < line_num_print)
    line_num_print--;
}

/* Plain cat. Copy the file behind 'input_desc' to STDOUT_FILENO.
BUF (of size BUFSIZE) is the I/O buffer, used by reads and writes.
Return true if successful. */

static bool
simple_cat (char *buf, idx_t bufsize)
{
/* Loop until the end of the file. */

while (true)
{
/* Read a block of input. */

size_t n_read = safe_read (input_desc, buf, bufsize);
if (n_read == SAFE_READ_ERROR)
{
    error (0, errno, "%s", quotef (infile));
    return false;
}

/* End of this file? */

if (n_read == 0)
    return true;

/* Write this block out. */

if (full_write (STDOUT_FILENO, buf, n_read) != n_read)
    write_error ();
}

/* Write any pending output to STDOUT_FILENO.
Pending is defined to be the *BPOUT - OUTBUF bytes starting at OUTBUF.
Then set *BPOUT to OUTPUT if it's not already that value. */

static inline void
write_pending (char *outbuf, char **bpout)
{
idx_t n_write = *bpout - outbuf;
if (0 < n_write)
{
    if (full_write (STDOUT_FILENO, outbuf, n_write) != n_write)
        write_error ();
    *bpout = outbuf;
}
}

```

```

        }

/* Copy the file behind 'input_desc' to STDOUT_FILENO.
Use INBUF and read INSIZE with each call,
and OUTBUF and write OUTSIZE with each call.
(The buffers are a bit larger than the I/O sizes.)
The remaining boolean args say what 'cat' options to use.

Return true if successful.
Called if any option more than -u was specified.

A newline character is always put at the end of the buffer, to make
an explicit test for buffer end unnecessary. */

static bool
cat(char *inbuf, idx_t insize, char *outbuf, idx_t outsize,
    bool show_nonprinting, bool show_tabs, bool number, bool number_nonblank,
    bool show_ends, bool squeeze_blank)
{
/* Last character read from the input buffer. */
unsigned char ch;

/* Determines how many consecutive newlines there have been in the
   input. 0 newlines makes NEWLINES -1, 1 newline makes NEWLINES 1,
   etc. Initially 0 to indicate that we are at the beginning of a
   new line. The "state" of the procedure is determined by
   NEWLINES. */
int newlines = newlines2;

#ifndef FIONREAD
/* If nonzero, use the FIONREAD ioctl, as an optimization.
   (On Ultrix, it is not supported on NFS file systems.) */
bool use_fionread = true;
#endif

/* The inbuf pointers are initialized so that BPIN > EOB, and thereby input
   is read immediately. */

/* Pointer to the first non-valid byte in the input buffer, i.e., the
   current end of the buffer. */
char *eob = inbuf;

/* Pointer to the next character in the input buffer. */
char *bpin = eob + 1;

/* Pointer to the position where the next character shall be written. */
char *bpout = outbuf;

while (true)
{
do
{
/* Write if there are at least OUTSIZE bytes in OUTBUF. */

if (outbuf + outsize <= bpout)
{
    char *wp = outbuf;
    idx_t remaining_bytes;
    do

```

```

{
    if (full_write (STDOUT_FILENO, wp, outsize) != outsize)
        write_error ();
    wp += outsize;
    remaining_bytes = bpout - wp;
}
while (outsize <= remaining_bytes);

/* Move the remaining bytes to the beginning of the
   buffer. */

memmove (outbuf, wp, remaining_bytes);
bpout = outbuf + remaining_bytes;
}

/* Is INBUF empty? */

if (bpin > eob)
{
    bool input_pending = false;
#endif FIONREAD
    int n_to_read = 0;

    /* Is there any input to read immediately?
       If not, we are about to wait,
       so write all buffered output before waiting. */

    if (use_fionread
        && ioctl (input_desc, FIONREAD, &n_to_read) < 0)
    {
        /* Ultrix returns EOPNOTSUPP on NFS;
           HP-UX returns ENOTTY on pipes.
           SunOS returns EINVAL and
           More/BSD returns ENODEV on special files
           like /dev/null.
           Irix-5 returns ENOSYS on pipes. */
        if (errno == EOPNOTSUPP || errno == ENOTTY
            || errno == EINVAL || errno == ENODEV
            || errno == ENOSYS)
            use_fionread = false;
        else
        {
            error (0, errno, _("cannot do ioctl on %s"),
                  quoteaf (infile));
            newlines2 = newlines;
            return false;
        }
    }
    if (n_to_read != 0)
        input_pending = true;
#endif
if (!input_pending)
    write_pending (outbuf, &bpout);

/* Read more input into INBUF. */

size_t n_read = safe_read (input_desc, inbuf, insize);
if (n_read == SAFE_READ_ERROR)
{

```

```

error (0, errno, "%s", quotef (infile));
write_pending (outbuf, &bpout);
newlines2 = newlines;
return false;
}
if (n_read == 0)
{
    write_pending (outbuf, &bpout);
    newlines2 = newlines;
    return true;
}

/* Update the pointers and insert a sentinel at the buffer
end. */

bpin = inbuf;
eob = bpin + n_read;
*eob = '\n';
}
else
{
    /* It was a real (not a sentinel) newline. */

    /* Was the last line empty?
       (i.e., have two or more consecutive newlines been read?) */

    if (++newlines > 0)
    {
        if (newlines >= 2)
        {
            /* Limit this to 2 here. Otherwise, with lots of
               consecutive newlines, the counter could wrap
               around at INT_MAX. */
            newlines = 2;

            /* Are multiple adjacent empty lines to be substituted
               by single ditto (-s), and this was the second empty
               line? */
            if (squeeze_blank)
            {
                ch = *bpin++;
                continue;
            }
        }
    }

    /* Are line numbers to be written at empty lines (-n)? */

    if (number && !number_nonblank)
    {
        next_line_num ();
        bpout = stpcpy (bpout, line_num_print);
    }
}

/* Output a currency symbol if requested (-e). */
if (show_ends)
{
    if (pending_cr)
    {
        *bpout++ = '^';

```

```

        *bpout++ = 'M';
        pending_cr = false;
    }
    *bpout++ = '$';
}

/* Output the newline. */

*bpout++ = '\n';
}
ch = *bpin++;
}
while (ch == '\n');

/* Here CH cannot contain a newline character. */

if (pending_cr)
{
    *bpout++ = '\r';
    pending_cr = false;
}

/* Are we at the beginning of a line, and line numbers are requested? */

if (newlines >= 0 && number)
{
    next_line_num ();
    bpout = stpcpy (bpout, line_num_print);
}

/* The loops below continue until a newline character is found,
   which means that the buffer is empty or that a proper newline
   has been found. */

/* If quoting, i.e., at least one of -v, -e, or -t specified,
   scan for chars that need conversion. */
if (show_nonprinting)
{
    while (true)
    {
        if (ch >= 32)
        {
            if (ch < 127)
                *bpout++ = ch;
            else if (ch == 127)
            {
                *bpout++ = '^';
                *bpout++ = '?';
            }
        }
        else
        {
            *bpout++ = 'M';
            *bpout++ = '-';
            if (ch >= 128 + 32)
            {
                if (ch < 128 + 127)
                    *bpout++ = ch - 128;
                else
                {
                    *bpout++ = '^';

```

```

        *bpout++ = '?';
    }
}
else
{
    *bpout++ = '^';
    *bpout++ = ch - 128 + 64;
}
}
}
else if (ch == '\t' && !show_tabs)
    *bpout++ = '\t';
else if (ch == '\n')
{
    newlines = -1;
    break;
}
else
{
    *bpout++ = '^';
    *bpout++ = ch + 64;
}

ch = *bpin++;
}
}
else
{
/* Not quoting, neither of -v, -e, or -t specified. */
while (true)
{
    if (ch == '\t' && show_tabs)
    {
        *bpout++ = '^';
        *bpout++ = ch + 64;
    }
    else if (ch != '\n')
    {
        if (ch == '\r' && *bpin == '\n' && show_ends)
        {
            if (bpin == eob)
                pending_cr = true;
            else
            {
                *bpout++ = '^';
                *bpout++ = 'M';
            }
        }
        else
            *bpout++ = ch;
    }
    else
    {
        newlines = -1;
        break;
    }

    ch = *bpin++;
}
}

```

```

        }

/* Copy data from input to output using copy_file_range if possible.
Return 1 if successful, 0 if ordinary read+write should be tried,
-1 if a serious problem has been diagnosed. */

static int
copy_cat (void)
{
/* Copy at most COPY_MAX bytes at a time; this is min
   (SSIZE_MAX, SIZE_MAX) truncated to a value that is
   surely aligned well. */
ssize_t copy_max = MIN (SSIZE_MAX, SIZE_MAX) >> 30 << 30;

/* copy_file_range does not support some cases, and it
   incorrectly returns 0 when reading from the proc file
   system on the Linux kernel through at least 5.6.19 (2020),
   so fall back on read+write if the copy_file_range is
   unsupported or the input file seems empty. */

for (bool some_copied = false; ; some_copied = true)
    switch (copy_file_range (input_desc, nullptr, STDOUT_FILENO, nullptr,
                           copy_max, 0))
    {
    case 0:
        return some_copied;

    case -1:
        if (errno == ENOSYS || is_ENOTSUP (errno) || errno == EINVAL
            || errno == EBADF || errno == EXDEV || errno == ETXTBSY
            || errno == EPERM)
            return 0;
        error (0, errno, "%s", quotef (infile));
        return -1;
    }
}

#endif __MVS__
#define MVS_CAT "/bin/cat"
#define MVS_CAT_LEN (sizeof(MVS_CAT)-1)
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
static bool
do_mvs_cat (int argc, char **argv)
{
    pid_t pid = fork();
    if (pid == 0) {
        argv[0] = MVS_CAT;
        execvp(MVS_CAT, argv);
    } else if (pid > 0) {
        int wstatus;
        pid_t waitchild = waitpid(pid, &wstatus, WUNTRACED | WCONTINUED);
        if (waitchild == -1) {
            perror("waitpid");
            return false;
        }
        if (WIFEXITED(wstatus)) {

```

```

        return WEXITSTATUS(wstatus) == 0 ? true : false;
    } else if (WIFSIGNALED(wstatus)) {
        fprintf(stderr, "killed by signal %d\n", WTERMSIG(wstatus));
    }
}
#endif

int
main (int argc, char **argv)
{
/* Nonzero if we have ever read standard input. */
bool have_read_stdin = false;

struct stat stat_buf;

/* Variables that are set according to the specified options. */
bool number = false;
bool number_nonblank = false;
bool squeeze_blank = false;
bool show_ends = false;
bool show_nonprinting = false;
bool show_tabs = false;
int file_open_mode = O_RDONLY;

static struct option const long_options[] =
{
    {"number-nonblank", no_argument, nullptr, 'b'},
    {"number", no_argument, nullptr, 'n'},
    {"squeeze-blank", no_argument, nullptr, 's'},
    {"show-nonprinting", no_argument, nullptr, 'v'},
    {"show-ends", no_argument, nullptr, 'E'},
    {"show-tabs", no_argument, nullptr, 'T'},
    {"show-all", no_argument, nullptr, 'A'},
    {GETOPT_HELP_OPTION_DECL},
    {GETOPT_VERSION_OPTION_DECL},
    {nullptr, 0, nullptr, 0}
};

#endif __MVS__
{
    int i;

    for (i=0; i<argc; ++i) {
        if (argv[i][0] == '/' && argv[i][1] == '/') {
            bool mvsok = do_mvs_cat(argc, argv);
            main_exit (mvsok ? EXIT_SUCCESS : EXIT_FAILURE);
        }
    }
}

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

/* Arrange to close stdout if we exit via the
   case_GETOPT_HELP_CHAR or case_GETOPT_VERSION_CHAR code.

```

```

Normally STDOUT_FILENO is used rather than stdout, so
close_stdout does nothing. */
atexit (close_stdout);

/* Parse command line options. */

int c;
while ((c = getopt_long (argc, argv, "benstuvAET", long_options, nullptr))
      != -1)
{
  switch (c)
  {
    case 'b':
      number = true;
      number_nonblank = true;
      break;

    case 'e':
      show_ends = true;
      show_nonprinting = true;
      break;

    case 'n':
      number = true;
      break;

    case 's':
      squeeze_blank = true;
      break;

    case 't':
      show_tabs = true;
      show_nonprinting = true;
      break;

    case 'u':
      /* We provide the -u feature unconditionally. */
      break;

    case 'v':
      show_nonprinting = true;
      break;

    case 'A':
      show_nonprinting = true;
      show_ends = true;
      show_tabs = true;
      break;

    case 'E':
      show_ends = true;
      break;

    case 'T':
      show_tabs = true;
      break;

    case_GETOPT_HELP_CHAR;
    case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);
  }
}

```

```

        default:
        usage (EXIT_FAILURE);
    }
}

/* Get device, i-node number, and optimal blocksize of output. */

if (fstat (STDOUT_FILENO, &stat_buf) < 0)
    error (EXIT_FAILURE, errno, _("standard output"));

/* Optimal size of i/o operations of output. */
idx_t outsize = io_blksize (&stat_buf);

/* Device and I-node number of the output. */
dev_t out_dev = stat_buf.st_dev;
ino_t out_ino = stat_buf.st_ino;

/* True if the output is a regular file. */
bool out_isreg = S_ISREG (stat_buf.st_mode) != 0;

if (! (number || show_ends || squeeze_blank))
{
    file_open_mode |= O_BINARY;
    xset_binary_mode (STDOUT_FILENO, O_BINARY);
}

/* Main loop. */

infile = "-";
int argind = optind;
bool ok = true;
idx_t page_size = getpagesize ();

do
{
    if (argind < argc)
        infile = argv[argind];

    bool reading_stdin = STREQ (infile, "-");
    if (reading_stdin)
    {
        have_read_stdin = true;
        input_desc = STDIN_FILENO;
        if (file_open_mode & O_BINARY)
            xset_binary_mode (STDIN_FILENO, O_BINARY);
    }
    else
    {
        input_desc = open (infile, file_open_mode);
        if (input_desc < 0)
        {
            error (0, errno, "%s", quotef (infile));
            ok = false;
            continue;
        }
    }

    if (fstat (input_desc, &stat_buf) < 0)
    {

```

```

error (0, errno, "%s", quotef (infile));
ok = false;
goto contin;
}

/* Optimal size of i/o operations of input. */
idx_t insize = io_blksize (&stat_buf);

fdadvise (input_desc, 0, 0, FADVISE_SEQUENTIAL);

/* Don't copy a nonempty regular file to itself, as that would
merely exhaust the output device. It's better to catch this
error earlier rather than later. */

if (out_isreg
    && stat_buf.st_dev == out_dev && stat_buf.st_ino == out_ino
    && lseek (input_desc, 0, SEEK_CUR) < stat_buf.st_size)
{
    error (0, 0, _("'%s: input file is output file"), quotef (infile));
    ok = false;
    goto contin;
}

/* Pointer to the input buffer. */
char *inbuf;

/* Select which version of 'cat' to use. If any format-oriented
options were given use 'cat'; if not, use 'copy_cat' if it
works, 'simple_cat' otherwise. */

if (! (number || show_ends || show_nonprinting
       || show_tabs || squeeze_blank))
{
    int copy_cat_status =
        out_isreg && S_ISREG (stat_buf.st_mode) ? copy_cat () : 0;
    if (copy_cat_status != 0)
    {
        inbuf = nullptr;
        ok &= 0 < copy_cat_status;
    }
    else
    {
        insize = MAX (insize, outsize);
        inbuf = xalignalloc (page_size, insize);
        ok &= simple_cat (inbuf, insize);
    }
}
else
{
    /* Allocate, with an extra byte for a newline sentinel. */
    inbuf = xalignalloc (page_size, insize + 1);
}

/* Why are
(OUTSIZE - 1 + INSIZE * 4 + LINE_COUNTER_BUF_LEN)
bytes allocated for the output buffer?

```

A test whether output needs to be written is done when the input buffer empties or when a newline appears in the input. After output is written, at most (OUTSIZE - 1) bytes will remain in the buffer. Now INSIZE bytes of input is read. Each input character

may grow by a factor of 4 (by the prepending of M-^). If all characters do, and no newlines appear in this block of input, we will have at most (OUTSIZE - 1 + INSIZE \* 4) bytes in the buffer. If the last character in the preceding block of input was a newline, a line number may be written (according to the given options) as the first thing in the output buffer. (Done after the new input is read, but before processing of the input begins.) A line number requires seldom more than LINE\_COUNTER\_BUF\_LEN positions.

Align the output buffer to a page size boundary, for efficiency on some paging implementations. \*/

```

idx_t bufsize;
if (ckd_mul (&bufsize, insize, 4)
    || ckd_add (&bufsize, bufsize, outsize)
    || ckd_add (&bufsize, bufsize, LINE_COUNTER_BUF_LEN - 1))
    xalloc_die ();
char *outbuf = xalignalloc (page_size, bufsize);

ok &= cat (inbuf, insize, outbuf, outsize, show_nonprinting,
            show_tabs, number, number_nonblank, show_ends,
            squeeze_blank);

alignfree (outbuf);
}

alignfree (inbuf);

contin:
if (!reading_stdin && close (input_desc) < 0)
{
    error (0, errno, "%s", quotef (infile));
    ok = false;
}
while (++argind < argc);

if (pending_cr)
{
    if (full_write (STDOUT_FILENO, "\r", 1) != 1)
        write_error ();
}

if (have_read_stdin && close (STDIN_FILENO) < 0)
    error (EXIT_FAILURE, errno, _("closing standard input"));

return ok ? EXIT_SUCCESS : EXIT_FAILURE;
}
"""

,
"patch":
"""

diff --git a/src/cat.c b/src/cat.c
index ac39a48..53bad79 100644
--- a/src/cat.c
+++ b/src/cat.c
@@ -532,6 +532,35 @@ copy_cat (void)
}
}
}
```

```

+ifdef __MVS__
+#define MVS_CAT "/bin/cat"
#define MVS_CAT_LEN (sizeof(MVS_CAT)-1)
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
static bool
do_mvs_cat (int argc, char **argv)
{
+
+ pid_t pid = fork();
+ if (pid == 0) {
+   argv[0] = MVS_CAT;
+   execvp(MVS_CAT, argv);
+ } else if (pid > 0) {
+   int wstatus;
+   pid_t waitchild = waitpid(pid, &wstatus, WUNTRACED | WCONTINUED);
+   if (waitchild == -1) {
+     perror("waitpid");
+     return false;
+   }
+   if (WIFEXITED(wstatus)) {
+     return WEXITSTATUS(wstatus) == 0 ? true : false;
+   } else if (WIFSIGNALED(wstatus)) {
+     fprintf(stderr, "killed by signal %d\n", WTERMSIG(wstatus));
+   }
+ }
+}
#endif

int
main (int argc, char **argv)
@@ -564,6 +593,19 @@ main (int argc, char **argv)
  {nullptr, 0, nullptr, 0}
};

#ifndef __MVS__
{
+ int i;
+
+ for (i=0; i<argc; ++i) {
+   if (argv[i][0] == '/' && argv[i][1] == '/') {
+     bool mvsok = do_mvs_cat(argc, argv);
+     main_exit (mvsok ? EXIT_SUCCESS : EXIT_FAILURE);
+   }
+ }
#endif
+
initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
"""
},
{
  "wrong_code": """
/* copy.c -- core functions for copying files and directories
Copyright (C) 1989-2024 Free Software Foundation, Inc.

```

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>. \*/

/\* Extracted from cp.c and librarified by Jim Meyering. \*/

```
#include <config.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <selinux/selinux.h>

#if HAVE_HURD_H
# include <hurd.h>
#endif
#if HAVE_PRIV_H
# include <priv.h>
#endif

#include "system.h"
#include "acl.h"
#include "alignalloc.h"
#include "assure.h"
#include "backupfile.h"
#include "buffer-lcm.h"
#include "canonicalize.h"
#include "copy.h"
#include "cp-hash.h"
#include "fadvise.h"
#include "fcntl--.h"
#include "file-set.h"
#include "filemode.h"
#include "filenamecat.h"
#include "force-link.h"
#include "full-write.h"
#include "hash.h"
#include "hash-triple.h"
#include "ignore-value.h"
#include "ioblksize.h"
#include "quote.h"
#include "renameatu.h"
#include "root-uid.h"
#include "same.h"
#include "savedir.h"
#include "stat-size.h"
#include "stat-time.h"
#include "utimecmp.h"
#include "utimens.h"
#include "write-any-file.h"
#include "areadlink.h"
```

```

#include "yesno.h"
#include "selinux.h"

#ifndef USE_XATTR
# define USE_XATTR false
#endif

#if USE_XATTR
# include <attr/error_context.h>
# include <attr/libattr.h>
# include <stdarg.h>
# include "verror.h"
#endif

#if HAVE_LINUX_FALLOC_H
# include <linux/falloc.h>
#endif

/* See HAVE_FALLOCATE workaround when including this file. */
#ifndef HAVE_LINUX_FS_H
# include <linux/fs.h>
#endif

#ifndef FICLONE && defined __linux__
# define FICLONE _IOW(0x94, 9, int)
#endif

#ifndef HAVE_FCLONEFILEAT && !USE_XATTR
# include <sys/clonefile.h>
#endif

#ifndef USE_ACL
# define USE_ACL 0
#endif

#define SAME_OWNER(A, B) ((A).st_uid == (B).st_uid)
#define SAME_GROUP(A, B) ((A).st_gid == (B).st_gid)
#define SAME_OWNER_AND_GROUP(A, B) (SAME_OWNER(A, B) && SAME_GROUP(A,
B))

/* LINK_FOLLOWING_SYMLINKS is tri-state; if it is -1, we don't know
how link() behaves, so assume we can't hardlink symlinks in that case. */
#ifndef (defined HAVE_LINKAT && !LINKAT_SYMLINK_NOTSUP) || !
LINK_FOLLOWING_SYMLINKS
# define CAN_HARDLINK_SYMLINKS 1
#else
# define CAN_HARDLINK_SYMLINKS 0
#endif

struct dir_list
{
    struct dir_list *parent;
    ino_t ino;
    dev_t dev;
};

/* Initial size of the cp.dest_info hash table. */
#define DEST_INFO_INITIAL_CAPACITY 61

static bool copy_internal (char const *src_name, char const *dst_name,

```

```

        int dst_dirfd, char const *dst_relname,
        int nonexistent_dst, struct stat const *parent,
        struct dir_list *ancestors,
        const struct cp_options *x,
        bool command_line_arg,
        bool *first_dir_created_per_command_line_arg,
        bool *copy_into_self,
        bool *rename_succeeded);
static bool owner_failure_ok (struct cp_options const *x);

/* Pointers to the file names: they're used in the diagnostic that is issued
when we detect the user is trying to copy a directory into itself. */
static char const *top_level_src_name;
static char const *top_level_dst_name;

enum copy_debug_val
{
COPY_DEBUG_UNKNOWN,
COPY_DEBUG_NO,
COPY_DEBUG_YES,
COPY_DEBUG_EXTERNAL,
COPY_DEBUG_EXTERNAL_INTERNAL,
COPY_DEBUG_AVOIDED,
COPY_DEBUG_UNSUPPORTED,
};

/* debug info about the last file copy. */
static struct copy_debug
{
enum copy_debug_val offload;
enum copy_debug_val reflink;
enum copy_debug_val sparse_detection;
} copy_debug;

static const char*
copy_debug_string (enum copy_debug_val debug_val)
{
switch (debug_val)
{
case COPY_DEBUG_NO: return "no";
case COPY_DEBUG_YES: return "yes";
case COPY_DEBUG_AVOIDED: return "avoided";
case COPY_DEBUG_UNSUPPORTED: return "unsupported";
default: return "unknown";
}
}

static const char*
copy_debug_sparse_string (enum copy_debug_val debug_val)
{
switch (debug_val)
{
case COPY_DEBUG_NO: return "no";
case COPY_DEBUG_YES: return "zeros";
case COPY_DEBUG_EXTERNAL: return "SEEK_HOLE";
case COPY_DEBUG_EXTERNAL_INTERNAL: return "SEEK_HOLE + zeros";
default: return "unknown";
}
}

```

```

/* Print --debug output on standard output. */
static void
emit_debug (const struct cp_options *x)
{
if (!x->hard_link && !x->symbolic_link && x->data_copy_required)
    printf ("copy offload: %s, reflink: %s, sparse detection: %s\n",
            copy_debug_string (copy_debug.offload),
            copy_debug_string (copy_debug.reflink),
            copy_debug_sparse_string (copy_debug.sparse_detection));
}

#ifndef DEV_FD_MIGHT_BE_CHR
#define DEV_FD_MIGHT_BE_CHR false
#endif

/* Act like fstat (DIRFD, FILENAME, ST, FLAGS), except when following
symbolic links on Solaris-like systems, treat any character-special
device like /dev/fd/0 as if it were the file it is open on. */
static int
follow_fstatat (int dirfd, char const *filename, struct stat *st, int flags)
{
int result = fstatat (dirfd, filename, st, flags);

if (DEV_FD_MIGHT_BE_CHR && result == 0 && !(flags & AT_SYMLINK_NOFOLLOW)
    && S_ISCHR (st->st_mode))
{
    static dev_t stdin_rdev;
    static signed char stdin_rdev_status;
    if (stdin_rdev_status == 0)
    {
        struct stat stdin_st;
        if (stat ("/dev/stdin", &stdin_st) == 0 && S_ISCHR (stdin_st.st_mode)
            && minor (stdin_st.st_rdev) == STDIN_FILENO)
        {
            stdin_rdev = stdin_st.st_rdev;
            stdin_rdev_status = 1;
        }
        else
            stdin_rdev_status = -1;
    }
    if (0 < stdin_rdev_status && major (stdin_rdev) == major (st->st_rdev))
        result = fstat (minor (st->st_rdev), st);
}
}

return result;
}

/* Attempt to punch a hole to avoid any permanent
speculative preallocation on file systems such as XFS.
Return values as per fallocate(2) except ENOSYS etc. are ignored. */

static int
punch_hole (int fd, off_t offset, off_t length)
{
int ret = 0;
/* +0 is to work around older <linux/fs.h> defining HAVE_FALLOCATE to empty. */
#if HAVE_FALLOCATE + 0
# if defined FALLOC_FL_PUNCH_HOLE && defined FALLOC_FL_KEEP_SIZE
ret = fallocate (fd, FALLOC_FL_PUNCH_HOLE | FALLOC_FL_KEEP_SIZE,
                offset, length);

```

```

if (ret < 0 && (is_ENOTSUP (errno) || errno == ENOSYS))
    ret = 0;
#endif
#endif
return ret;
}

/* Create a hole at the end of a file,
   avoiding preallocation if requested. */

static bool
create_hole (int fd, char const *name, bool punch_holes, off_t size)
{
off_t file_end = lseek (fd, size, SEEK_CUR);

if (file_end < 0)
{
    error (0, errno, _("cannot lseek %s"), quoteaf (name));
    return false;
}

/* Some file systems (like XFS) preallocate when write extending a file.
   I.e., a previous write() may have preallocated extra space
   that the seek above will not discard. A subsequent write() could
   then make this allocation permanent. */
if (punch_holes && punch_hole (fd, file_end - size, size) < 0)
{
    error (0, errno, _("error deallocating %s"), quoteaf (name));
    return false;
}

return true;
}

/* Whether an errno value ERR, set by FICLONE or copy_file_range,
   indicates that the copying operation has terminally failed, even
   though it was invoked correctly (so that, e.g, EBADF cannot occur)
   and even though !is_CLONENOTSUP (ERR). */

static bool
is_terminal_error (int err)
{
return err == EIO || err == ENOMEM || err == ENOSPC || err == EDQUOT;
}

/* Similarly, whether ERR indicates that the copying operation is not
   supported or allowed for this file or process, even though the
   operation was invoked correctly. */

static bool
is_CLONENOTSUP (int err)
{
return err == ENOSYS || err == ENOTTY || is_ENOTSUP (err)
    || err == EINVAL || err == EBADF
    || err == EXDEV || err == ETXTBSY
    || err == EPERM || err == EACCES;
}

```

```

/* Copy the regular file open on SRC_FD/SRC_NAME to DST_FD/DST_NAME,
honoring the MAKE_HOLES setting and using the BUF_SIZE-byte buffer
*ABUF for temporary storage, allocating it lazily if *ABUF is null.
Copy no more than MAX_N_READ bytes.
Return true upon successful completion;
print a diagnostic and return false upon error.
Note that for best results, BUF should be "well"-aligned.
Set *LAST_WRITE_MADE_HOLE to true if the final operation on
DEST_FD introduced a hole. Set *TOTAL_N_READ to the number of
bytes read. */
static bool
sparse_copy (int src_fd, int dest_fd, char **abuf, size_t buf_size,
             size_t hole_size, bool punch_holes, bool allow_relink,
             char const *src_name, char const *dst_name,
             uintmax_t max_n_read, off_t *total_n_read,
             bool *last_write_made_hole)
{
    *last_write_made_hole = false;
    *total_n_read = 0;

    if (copy_debug.sparse_detection == COPY_DEBUG_UNKNOWN)
        copy_debug.sparse_detection = hole_size ? COPY_DEBUG_YES : COPY_DEBUG_NO;
    else if (hole_size && copy_debug.sparse_detection == COPY_DEBUG_EXTERNAL)
        copy_debug.sparse_detection = COPY_DEBUG_EXTERNAL_INTERNAL;

    /* If not looking for holes, use copy_file_range if functional,
       but don't use if reflink disallowed as that may be implicit. */
    if (!hole_size && allow_relink)
        while (max_n_read)
    {
        /* Copy at most COPY_MAX bytes at a time; this is min
           (SSIZE_MAX, SIZE_MAX) truncated to a value that is
           surely aligned well. */
        ssize_t copy_max = MIN (SSIZE_MAX, SIZE_MAX) >> 30 << 30;
        ssize_t n_copied = copy_file_range (src_fd, nullptr, dest_fd, nullptr,
                                           MIN (max_n_read, copy_max), 0);
        if (n_copied == 0)
        {
            /* copy_file_range incorrectly returns 0 when reading from
               the proc file system on the Linux kernel through at
               least 5.6.19 (2020), so fall back on 'read' if the
               input file seems empty. */
            if (*total_n_read == 0)
                break;
            copy_debug.offload = COPY_DEBUG_YES;
            return true;
        }
        if (n_copied < 0)
        {
            copy_debug.offload = COPY_DEBUG_UNSUPPORTED;

            /* Consider operation unsupported only if no data copied.
               For example, EPERM could occur if copy_file_range not enabled
               in seccomp filters, so retry with a standard copy. EPERM can
               also occur for immutable files, but that would only be in the
               edge case where the file is made immutable after creating,
               in which case the (more accurate) error is still shown. */
            if (*total_n_read == 0 && is_CLONENOTSUP (errno))
                break;
        }
    }
}

```

```

/* ENOENT was seen sometimes across CIFS shares, resulting in
no data being copied, but subsequent standard copies succeed. */
if (*total_n_read == 0 && errno == ENOENT)
break;

if (errno == EINTR)
n_copied = 0;
else
{
    error (0, errno, _("error copying %s to %s"),
        quoteaf_n (0, src_name), quoteaf_n (1, dst_name));
    return false;
}
copy_debug.offload = COPY_DEBUG_YES;
max_n_read -= n_copied;
*total_n_read += n_copied;
}
else
copy_debug.offload = COPY_DEBUG_AVOIDED;

bool make_hole = false;
off_t psize = 0;

while (max_n_read)
{
if (!*abuf)
*abuf = xalignalloc (getpagesize (), buf_size);
char *buf = *abuf;
ssize_t n_read = read (src_fd, buf, MIN (max_n_read, buf_size));
if (n_read < 0)
{
if (errno == EINTR)
continue;
error (0, errno, _("error reading %s"), quoteaf (src_name));
return false;
}
if (n_read == 0)
break;
max_n_read -= n_read;
*total_n_read += n_read;

/* Loop over the input buffer in chunks of hole_size. */
size_t csize = hole_size ? hole_size : buf_size;
char *cbuf = buf;
char *pbuf = buf;

while (n_read)
{
bool prev_hole = make_hole;
csize = MIN (csize, n_read);

if (hole_size && csize)
make_hole = is_nul (cbuf, csize);

bool transition = (make_hole != prev_hole) && psize;
bool last_chunk = (n_read == csize && ! make_hole) || ! csize;

if (transition || last_chunk)

```

```

{
if (! transition)
    psize += csizes;

if (! prev_hole)
{
    if (full_write (dest_fd, pbuf, psize) != psize)
    {
        error (0, errno, _("error writing %s"),
               quoteaf (dst_name));
        return false;
    }
}
else
{
    if (! create_hole (dest_fd, dst_name, punch_holes, psize))
        return false;
}

pbuf = cbuf;
psize = csizes;

if (last_chunk)
{
    if (! csizes)
        n_read = 0; /* Finished processing buffer. */

    if (transition)
        csizes = 0; /* Loop again to deal with last chunk. */
    else
        psize = 0; /* Reset for next read loop. */
}
else /* Coalesce writes/seeks. */
{
    if (ckd_add (&psize, psize, csizes))
    {
        error (0, 0, _("overflow reading %s"), quoteaf (src_name));
        return false;
    }
}

n_read -= csizes;
cbuf += csizes;
}

/*last_write_made_hole = make_hole;

/* It's tempting to break early here upon a short read from
   a regular file. That would save the final read syscall
   for each file. Unfortunately that doesn't work for
   certain files in /proc or /sys with linux kernels. */
}

/* Ensure a trailing hole is created, so that subsequent
   calls of sparse_copy() start at the correct offset. */
if (make_hole && ! create_hole (dest_fd, dst_name, punch_holes, psize))
    return false;
else
    return true;

```

```

}

/* Perform the O(1) btrfs clone operation, if possible.
Upon success, return 0. Otherwise, return -1 and set errno. */
static inline int
clone_file (int dest_fd, int src_fd)
{
#ifndef FICLONE
return ioctl (dest_fd, FICLONE, src_fd);
#else
(void) dest_fd;
(void) src_fd;
errno = ENOTSUP;
return -1;
#endif
}

/* Write N_BYTES zero bytes to file descriptor FD. Return true if successful.
Upon write failure, set errno and return false. */
static bool
write_zeros (int fd, off_t n_bytes)
{
static char *zeros;
static size_t nz = IO_BUFSIZE;

/* Attempt to use a relatively large calloc'd source buffer for
   efficiency, but if that allocation fails, resort to a smaller
   statically allocated one. */
if (zeros == nullptr)
{
    {
static char fallback[1024];
zeros = calloc (nz, 1);
if (zeros == nullptr)
    {
zeros = fallback;
nz = sizeof fallback;
    }
}
}

while (n_bytes)
{
size_t n = MIN (nz, n_bytes);
if ((full_write (fd, zeros, n)) != n)
    return false;
n_bytes -= n;
}

return true;
}

#endif /* _BTRFS_H */

#endif /* _LIBBTRFS_H */

```

```

Return true if successful, false (with a diagnostic) otherwise. */

static bool
lseek_copy (int src_fd, int dest_fd, char **abuf, size_t buf_size,
            size_t hole_size, off_t ext_start, off_t src_total_size,
            enum Sparse_type sparse_mode,
            bool allow_relink,
            char const *src_name, char const *dst_name)
{
off_t last_ext_start = 0;
off_t last_ext_len = 0;
off_t dest_pos = 0;
bool wrote_hole_at_eof = true;

copy_debug.sparse_detection = COPY_DEBUG_EXTERNAL;

while (0 <= ext_start)
{
off_t ext_end = lseek (src_fd, ext_start, SEEK_HOLE);
if (ext_end < 0)
{
if (errno != ENXIO)
    goto cannot_lseek;
ext_end = src_total_size;
if (ext_end <= ext_start)
{
/* The input file grew; get its current size. */
src_total_size = lseek (src_fd, 0, SEEK_END);
if (src_total_size < 0)
    goto cannot_lseek;

/* If the input file shrank after growing, stop copying. */
if (src_total_size <= ext_start)
    break;

ext_end = src_total_size;
}
}
/* If the input file must have grown, increase its measured size. */
if (src_total_size < ext_end)
    src_total_size = ext_end;

if (lseek (src_fd, ext_start, SEEK_SET) < 0)
    goto cannot_lseek;

wrote_hole_at_eof = false;
off_t ext_hole_size = ext_start - last_ext_start - last_ext_len;

if (ext_hole_size)
{
if (sparse_mode != SPARSE_NEVER)
{
if (! create_hole (dest_fd, dst_name,
                  sparse_mode == SPARSE_ALWAYS,
                  ext_hole_size))
    return false;
wrote_hole_at_eof = true;
}
else
{

```

```

/* When not inducing holes and when there is a hole between
   the end of the previous extent and the beginning of the
   current one, write zeros to the destination file. */
if (! write_zeros (dest_fd, ext_hole_size))
{
{
    error (0, errno, _("'%s: write failed"),
          quoteaf (dst_name));
    return false;
}
}

off_t ext_len = ext_end - ext_start;
last_ext_start = ext_start;
last_ext_len = ext_len;

/* Copy this extent, looking for further opportunities to not
   bother to write zeros if --sparse=always, since SEEK_HOLE
   is conservative and may miss some holes. */
off_t n_read;
bool read_hole;
if (! sparse_copy (src_fd, dest_fd, abuf, buf_size,
                  sparse_mode != SPARSE_ALWAYS ? 0 : hole_size,
                  true, allow_relink, src_name, dst_name,
                  ext_len, &n_read, &read_hole))
    return false;

dest_pos = ext_start + n_read;
if (n_read)
    wrote_hole_at_eof = read_hole;
if (n_read < ext_len)
{
{
    /* The input file shrank. */
    src_total_size = dest_pos;
    break;
}

ext_start = lseek (src_fd, dest_pos, SEEK_DATA);
if (ext_start < 0 && errno != ENXIO)
    goto cannot_lseek;
}

/* When the source file ends with a hole, we have to do a little more work,
   since the above copied only up to and including the final extent.
   In order to complete the copy, we may have to insert a hole or write
   zeros in the destination corresponding to the source file's hole-at-EOF.

   In addition, if the final extent was a block of zeros at EOF and we've
   just converted them to a hole in the destination, we must call ftruncate
   here in order to record the proper length in the destination. */
if ((dest_pos < src_total_size || wrote_hole_at_eof)
    && ! (sparse_mode == SPARSE_NEVER
          ? write_zeros (dest_fd, src_total_size - dest_pos)
          : ftruncate (dest_fd, src_total_size) == 0))
{
{
    error (0, errno, _("failed to extend %s"), quoteaf (dst_name));
    return false;
}

if (sparse_mode == SPARSE_ALWAYS && dest_pos < src_total_size

```

```

    && punch_hole (dest_fd, dest_pos, src_total_size - dest_pos) < 0)
{
    error (0, errno, _("error deallocating %s"), quoteaf (dst_name));
    return false;
}

return true;

cannot_lseek:
error (0, errno, _("cannot lseek %s"), quoteaf (src_name));
return false;
}
#endif

/* FIXME: describe */
/* FIXME: rewrite this to use a hash table so we avoid the quadratic
   performance hit that's probably noticeable only on trees deeper
   than a few hundred levels. See use of active_dir_map in remove.c */

ATTRIBUTE_PURE
static bool
is_ancestor (const struct stat *sb, const struct dir_list *ancestors)
{
while (ancestors != 0)
{
    if (ancestors->ino == sb->st_ino && ancestors->dev == sb->st_dev)
        return true;
    ancestors = ancestors->parent;
}
return false;
}

static bool
errno_unsupported (int err)
{
return err == ENOTSUP || err == ENODATA;
}

#if USE_XATTR
ATTRIBUTE_FORMAT ((printf, 2, 3))
static void
copy_attr_error (MAYBE_UNUSED struct error_context *ctx,
                 char const *fmt, ...)
{
if (!errno_unsupported (errno))
{
    int err = errno;
    va_list ap;

    /* use verror module to print error message */
    va_start (ap, fmt);
    verror (0, err, fmt, ap);
    va_end (ap);
}
}

ATTRIBUTE_FORMAT ((printf, 2, 3))
static void
copy_attr_allerror (MAYBE_UNUSED struct error_context *ctx,
                    char const *fmt, ...)

```

```

{
int err = errno;
va_list ap;

/* use perror module to print error message */
va_start (ap, fmt);
perror (0, err, fmt, ap);
va_end (ap);
}

static char const *
copy_attr_quote (MAYBE_UNUSED struct error_context *ctx, char const *str)
{
return quoteaf (str);
}

static void
copy_attr_free (MAYBE_UNUSED struct error_context *ctx,
               MAYBE_UNUSED char const *str)
{
}

/* Exclude SELinux extended attributes that are otherwise handled,
and are problematic to copy again. Also honor attributes
configured for exclusion in /etc/xattr.conf.
FIXME: Should we handle POSIX ACLs similarly?
Return zero to skip. */
static int
check_selinux_attr (char const *name, struct error_context *ctx)
{
return STRNCMP_LIT (name, "security.selinux")
    && attr_copy_check_permissions (name, ctx);
}

/* If positive SRC_FD and DST_FD descriptors are passed,
then copy by fd, otherwise copy by name. */

static bool
copy_attr (char const *src_path, int src_fd,
           char const *dst_path, int dst_fd, struct cp_options const *x)
{
bool all_errors = (!x->data_copy_required || x->require_preserve_xattr);
bool some_errors = (!all_errors && !x->reduce_diagnostics);
int (*check) (char const *, struct error_context *)
    = (x->preserve_security_context || x->set_security_context
       ? check_selinux_attr : nullptr);

# if 4 < __GNUC__ + (8 <= __GNUC_MINOR__)
/* Pacify gcc -Wsuggest-attribute=format through at least GCC 13.2.1. */
# pragma GCC diagnostic push
# pragma GCC diagnostic ignored "-Wsuggest-attribute=format"
# endif
struct error_context *ctx
    = (all_errors || some_errors
       ? (&(struct error_context) {
           .error = all_errors ? copy_attr_allerror : copy_attr_error,
           .quote = copy_attr_quote,
           .quote_free = copy_attr_free
         })
       : nullptr);

```

```

# if 4 < __GNUC__ + (8 <= __GNUC_MINOR__)
# pragma GCC diagnostic pop
# endif

return ! (0 <= src_fd && 0 <= dst_fd
    ? attr_copy_fd (src_path, src_fd, dst_path, dst_fd, check, ctx)
    : attr_copy_file (src_path, dst_path, check, ctx));
}
#endif /* USE_XATTR */

static bool
copy_attr (MAYBE_UNUSED char const *src_path,
    MAYBE_UNUSED int src_fd,
    MAYBE_UNUSED char const *dst_path,
    MAYBE_UNUSED int dst_fd,
    MAYBE_UNUSED struct cp_options const *x)
{
    return true;
}
#endif /* USE_XATTR */

/* Read the contents of the directory SRC_NAME_IN, and recursively
copy the contents to DST_NAME_IN aka DST_DIRFD+DST_RELNAME_IN.
NEW_DST is true if DST_NAME_IN is a directory
that was created previously in the recursion.
SRC_SB and ANCESTORS describe SRC_NAME_IN.
Set *COPY_INTO_SELF if SRC_NAME_IN is a parent of
(or the same as) DST_NAME_IN; otherwise, clear it.
Propagate *FIRST_DIR_CREATED_PER_COMMAND_LINE_ARG from
caller to each invocation of copy_internal. Be careful to
pass the address of a temporary, and to update
*FIRST_DIR_CREATED_PER_COMMAND_LINE_ARG only upon completion.
Return true if successful. */

static bool
copy_dir (char const *src_name_in, char const *dst_name_in,
    int dst_dirfd, char const *dst_relname_in, bool new_dst,
    const struct stat *src_sb, struct dir_list *ancestors,
    const struct cp_options *x,
    bool *first_dir_created_per_command_line_arg,
    bool *copy_into_self)
{
    char *name_space;
    char *namep;
    struct cp_options non_command_line_options = *x;
    bool ok = true;

    name_space = savedir (src_name_in, SAVEDIR_SORT_FASTREAD);
    if (name_space == nullptr)
    {
        /* This diagnostic is a bit vague because savedir can fail in
           several different ways. */
        error (0, errno, _("cannot access %s"), quoteaf (src_name_in));
        return false;
    }

    /* For cp's -H option, dereference command line arguments, but do not
       dereference symlinks that are found via recursive traversal. */
    if (x->dereference == DEREF_COMMAND_LINE_ARGUMENTS)
        non_command_line_options.dereference = DEREF_NEVER;

```

```

bool new_first_dir_created = false;
namep = name_space;
while (*namep != '\0')
{
    bool local_copy_into_self;
    char *src_name = file_name_concat (src_name_in, namep, nullptr);
    char *dst_name = file_name_concat (dst_name_in, namep, nullptr);
    bool first_dir_created = *first_dir_created_per_command_line_arg;
    bool rename_succeeded;

    ok &= copy_internal (src_name, dst_name, dst_dirfd,
                         dst_name + (dst_relname_in - dst_name_in),
                         new_dst, src_sb,
                         ancestors, &non_command_line_options, false,
                         &first_dir_created,
                         &local_copy_into_self, &rename_succeeded);
    *copy_into_self |= local_copy_into_self;

    free (dst_name);
    free (src_name);

    /* If we're copying into self, there's no point in continuing,
       and in fact, that would even infloop, now that we record only
       the first created directory per command line argument. */
    if (local_copy_into_self)
        break;

    new_first_dir_created |= first_dir_created;
    namep += strlen (namep) + 1;
}
free (name_space);
*first_dir_created_per_command_line_arg = new_first_dir_created;

return ok;
}

/* Change the file mode bits of the file identified by DESC or
DIRFD+NAME to MODE. Use DESC if DESC is valid and fchmod is
available, DIRFD+NAME otherwise. */

static int
fchmod_or_lchmod (int desc, int dirfd, char const *name, mode_t mode)
{
#if HAVE_FCHMOD
if (0 <= desc)
    return fchmod (desc, mode);
#endif
return lchmodat (dirfd, name, mode);
}

/* Change the ownership of the file identified by DESC or
DIRFD+NAME to UID+GID. Use DESC if DESC is valid and fchown is
available, DIRFD+NAME otherwise. */

static int
fchown_or_lchown (int desc, int dirfd, char const *name, uid_t uid, gid_t gid)
{
#if HAVE_FCHOWN
if (0 <= desc)

```

```

        return fchown (desc, uid, gid);
#endif
return lchownat (dirfd, name, uid, gid);
}

/* Set the owner and owning group of DEST_DESC to the st_uid and
st_gid fields of SRC_SB. If DEST_DESC is undefined (-1), set
the owner and owning group of DST_NAME aka DST_DIRFD+DST_RELNAME
instead; for safety prefer lchownat since no
symbolic links should be involved. DEST_DESC must
refer to the same file as DST_NAME if defined.
Upon failure to set both UID and GID, try to set only the GID.
NEW_DST is true if the file was newly created; otherwise,
DST_SB is the status of the destination.
Return 1 if the initial syscall succeeds, 0 if it fails but it's OK
not to preserve ownership, -1 otherwise. */

static int
set_owner (const struct cp_options *x, char const *dst_name,
           int dst_dirfd, char const *dst_relname, int dest_desc,
           struct stat const *src_sb, bool new_dst,
           struct stat const *dst_sb)
{
uid_t uid = src_sb->st_uid;
gid_t gid = src_sb->st_gid;

/* Naively changing the ownership of an already-existing file before
changing its permissions would create a window of vulnerability if
the file's old permissions are too generous for the new owner and
group. Avoid the window by first changing to a restrictive
temporary mode if necessary. */

if (!new_dst && (x->preserve_mode || x->move_mode || x->set_mode))
{
mode_t old_mode = dst_sb->st_mode;
mode_t new_mode =
    (x->preserve_mode || x->move_mode ? src_sb->st_mode : x->mode);
mode_t restrictive_temp_mode = old_mode & new_mode & S_IRWXU;

if ((USE_ACL
     || (old_mode & CHMOD_MODE_BITS
         & (~new_mode | S_ISUID | S_ISGID | S_ISVTX)))
     && qset_acl (dst_name, dest_desc, restrictive_temp_mode) != 0)
{
if (!owner_failure_ok (x))
    error (0, errno, _("clearing permissions for %s"),
           quoteaf (dst_name));
return -x->require_preserve;
}
}

if (fchown_or_lchown (dest_desc, dst_dirfd, dst_relname, uid, gid) == 0)
    return 1;

/* The ownership change failed. If the failure merely means we lack
privileges to change owner+group, try to change just the group
and ignore any failure of this. Otherwise, report an error. */
if (chown_failure_ok (x))
    ignore_value (fchown_or_lchown (dest_desc, dst_dirfd, dst_relname,
                                   -1, gid));

```

```

else
{
    error (0, errno, _("failed to preserve ownership for %s"),
           quoteaf (dst_name));
    if (x->require_preserve)
        return -1;
}

return 0;
}

/* Set the st_author field of DEST_DESC to the st_author field of
   SRC_SB. If DEST_DESC is undefined (-1), set the st_author field
   of DST_NAME instead. DEST_DESC must refer to the same file as
   DST_NAME if defined. */

static void
set_author (char const *dst_name, int dest_desc, const struct stat *src_sb)
{
#ifndef HAVE_STRUCT_STAT_ST_AUTHOR
/* FIXME: Modify the following code so that it does not
   follow symbolic links. */

/* Preserve the st_author field. */
file_t file = (dest_desc < 0
               ? file_name_lookup (dst_name, 0, 0)
               : getdport (dest_desc));
if (file == MACH_PORT_NULL)
    error (0, errno, _("failed to lookup file %s"), quoteaf (dst_name));
else
{
    error_t err = file_chauthor (file, src_sb->st_author);
    if (err)
        error (0, err, _("failed to preserve authorship for %s"),
               quoteaf (dst_name));
    mach_port_deallocate (mach_task_self (), file);
}
#endif
(void) dst_name;
(void) dest_desc;
(void) src_sb;
#endif
}

/* Set the default security context for the process. New files will
have this security context set. Also existing files can have their
context adjusted based on this process context, by
set_file_security_ctx() called with PROCESS_LOCAL=true.
This should be called before files are created so there is no race
where a file may be present without an appropriate security context.
Based on CP_OPTIONS, diagnose warnings and fail when appropriate.
Return FALSE on failure, TRUE on success. */

bool
set_process_security_ctx (char const *src_name, char const *dst_name,
                        mode_t mode, bool new_dst, const struct cp_options *x)
{
if (x->preserve_security_context)
{
    /* Set the default context for the process to match the source. */

```

```

bool all_errors = !x->data_copy_required || x->require_preserve_context;
bool some_errors = !all_errors && !x->reduce_diagnostics;
char *con_raw;

if (0 <= lgetfilecon_raw (src_name, &con_raw))
{
    if (setfscreatecon_raw (con_raw) < 0)
    {
        if (all_errors || (some_errors && !errno_unsupported (errno)))
            error (0, errno,
                   _("failed to set default file creation context to %s"),
                   quote (con_raw));
        if (x->require_preserve_context)
        {
            freecon (con_raw);
            return false;
        }
    }
    freecon (con_raw);
}
else
{
    if (all_errors || (some_errors && !errno_unsupported (errno)))
    {
        error (0, errno,
               _("failed to get security context of %s"),
               quoteaf (src_name));
    }
    if (x->require_preserve_context)
        return false;
}
else if (x->set_security_context)
{
    /* With -Z, adjust the default context for the process
       to have the type component adjusted as per the destination path. */
    if (new_dst && defaultcon (x->set_security_context, dst_name, mode) < 0
        && ! ignorable_ctx_err (errno))
    {
        error (0, errno,
               _("failed to set default file creation context for %s"),
               quoteaf (dst_name));
    }
}

return true;
}

/* Reset the security context of DST_NAME, to that already set
   as the process default if !X->set_security_context. Otherwise
   adjust the type component of DST_NAME's security context as
   per the system default for that path. Issue warnings upon
   failure, when allowed by various settings in X.
   Return false on failure, true on success. */

bool
set_file_security_ctx (char const *dst_name,
                      bool recurse, const struct cp_options *x)
{
bool all_errors = (!x->data_copy_required

```

```

    || x->require_preserve_context);
bool some_errors = !all_errors && !x->reduce_diagnostics;

if (! restorecon (x->set_security_context, dst_name, recurse))
{
    if (all_errors || (some_errors && !errno_unsupported (errno)))
        error (0, errno, _("failed to set the security context of %s"),
               quoteaf_n (0, dst_name));
    return false;
}

return true;
}

#ifndef HAVE_STRUCT_STAT_ST_BLOCKS
#define HAVE_STRUCT_STAT_ST_BLOCKS 0
#endif

/* Type of scan being done on the input when looking for sparseness. */
enum scantype
{
/* An error was found when determining scantype. */
ERROR_SCANTYPE,
/* No fancy scanning; just read and write. */
PLAIN_SCANTYPE,
/* Read and examine data looking for zero blocks; useful when
   attempting to create sparse output. */
ZERO_SCANTYPE,
/* lseek information is available. */
LSEEK_SCANTYPE,
};

/* Result of infer_scantype. */
union scan_inference
{
/* Used if infer_scantype returns LSEEK_SCANTYPE. This is the
   offset of the first data block, or -1 if the file has no data. */
off_t ext_start;
};

/* Return how to scan a file with descriptor FD and stat buffer SB.
*SCAN_INFERENCE is set to a valid value if returning LSEEK_SCANTYPE. */
static enum scantype
infer_scantype (int fd, struct stat const *sb,
                union scan_inference *scan_inference)
{
scan_inference->ext_start = -1; /* avoid -Wmaybe-uninitialized */

/* Only attempt SEEK_HOLE if this heuristic
   suggests the file is sparse. */
if (! (HAVE_STRUCT_STAT_ST_BLOCKS
      && S_ISREG (sb->st_mode)
      && STP_NBLOCKS (sb) < sb->st_size / ST_NBLOCKSIZE))
    return PLAIN_SCANTYPE;

#ifndef SEEK_HOLE
off_t ext_start = lseek (fd, 0, SEEK_DATA);

```

```

if (0 <= ext_start || errno == ENXIO)
{
    scan_inference->ext_start = ext_start;
    return LSEEK_SCANTYPE;
}
else if (errno != EINVAL && !is_ENOTSUP (errno))
    return ERROR_SCANTYPE;
#endif

return ZERO_SCANTYPE;
}

#if HAVE_FCLONEFILEAT && !USE_XATTR
# include <sys/acl.h>
/* Return true if FD has a nontrivial ACL. */
static bool
fd_has_acl (int fd)
{
/* Every platform with fclonefileat (macOS 10.12 or later) also has
   acl_get_fd_np. */
bool has_acl = false;
acl_t acl = acl_get_fd_np (fd, ACL_TYPE_EXTENDED);
if (acl)
{
    acl_entry_t ace;
    has_acl = 0 <= acl_get_entry (acl, ACL_FIRST_ENTRY, &ace);
    acl_free (acl);
}
return has_acl;
}
#endif

/* Handle failure from FICLONE or fclonefileat.
Return FALSE if it's a terminal failure for this file. */

static bool
handle_clone_fail (int dst_dirfd, char const *dst_relname,
                  char const *src_name, char const *dst_name,
                  int dest_desc, bool new_dst, enum Reflink_type reflink_mode)
{
/* When the clone operation fails, report failure only with errno values
   known to mean trouble when the clone is supported and called properly.
   Do not report failure merely because !is_CLONENOTSUP (errno),
   as systems may yield oddball errno values here with FICLONE,
   and is_CLONENOTSUP is not appropriate for fclonefileat. */
bool report_failure = is_terminal_error (errno);

if (reflink_mode == REFLINK_ALWAYS || report_failure)
    error (0, errno, _("failed to clone %s from %s"),
           quoteaf_n (0, dst_name), quoteaf_n (1, src_name));

/* Remove the destination if cp --reflink=always created it
   but cloned no data. */
if (new_dst /* currently not for fclonefileat(). */
    && reflink_mode == REFLINK_ALWAYS
    && (! report_failure) || lseek (dest_desc, 0, SEEK_END) == 0
    && unlinkat (dst_dirfd, dst_relname, 0) != 0 && errno != ENOENT)
    error (0, errno, _("cannot remove %s"), quoteaf (dst_name));

if (! report_failure)

```

```

copy_debug.reflink = COPY_DEBUG_UNSUPPORTED;

if (reflink_mode == REFLINK_ALWAYS || report_failure)
    return false;

return true;
}

/* Copy a regular file from SRC_NAME to DST_NAME aka DST_DIRFD+DST_RELNAME.
If the source file contains holes, copies holes and blocks of zeros
in the source file as holes in the destination file.
(Holes are read as zeroes by the 'read' system call.)
When creating the destination, use DST_MODE & ~OMITTED_PERMISSIONS
as the third argument in the call to open, adding
OMITTED_PERMISSIONS after copying as needed.
X provides many option settings.
Return true if successful.
*NEW_DST is initially as in copy_internal.
If successful, set *NEW_DST to true if the destination file was created and
to false otherwise; if unsuccessful, perhaps set *NEW_DST to some value.
SRC_SB is the result of calling follow_fstatat on SRC_NAME;
it might be updated by calling fstat again on the same file,
to give it slightly more up-to-date contents. */

static bool
copy_reg (char const *src_name, char const *dst_name,
          int dst_dirfd, char const *dst_relname,
          const struct cp_options *x,
          mode_t dst_mode, mode_t omitted_permissions, bool *new_dst,
          struct stat *src_sb)
{
    char *buf = nullptr;
    int dest_desc;
    int dest_errno;
    int source_desc;
    mode_t extra_permissions;
    struct stat sb;
    struct stat src_open_sb;
    union scan_inference scan_inference;
    bool return_val = true;
    bool data_copy_required = x->data_copy_required;
    bool preserve_xattr = USE_XATTR & x->preserve_xattr;

    copy_debug.offload = COPY_DEBUG_UNKNOWN;
    copy_debug.reflink = x->reflink_mode ? COPY_DEBUG_UNKNOWN : COPY_DEBUG_NO;
    copy_debug.sparse_detection = COPY_DEBUG_UNKNOWN;

    source_desc = open (src_name,
                       (O_RDONLY | O_BINARY
                        | (x->dereference == DEREF_NEVER ? O_NOFOLLOW : 0)));
    if (source_desc < 0)
    {
        error (0, errno, _("cannot open %s for reading"), quoteaf (src_name));
        return false;
    }

    if (fstat (source_desc, &src_open_sb) != 0)
    {
        error (0, errno, _("cannot fstat %s"), quoteaf (src_name));

```

```

return_val = false;
goto close_src_desc;
}

/* Compare the source dev/ino from the open file to the incoming,
   saved ones obtained via a previous call to stat. */
if (!psame_inode (src_sb, &src_open_sb))
{
    error (0, 0,
           _("skipping file %s, as it was replaced while being copied"),
           quoteaf (src_name));
    return_val = false;
    goto close_src_desc;
}

/* Might as well tell the caller about the latest version of the
   source file status, since we have it already. */
*src_sb = src_open_sb;
mode_t src_mode = src_sb->st_mode;

/* The semantics of the following open calls are mandated
   by the specs for both cp and mv. */
if (!*new_dst)
{
    int open_flags =
        O_WRONLY | O_BINARY | (data_copy_required ? O_TRUNC : 0);
    dest_desc = openat (dst_dirfd, dst_relnname, open_flags);
    dest_errno = errno;

    /* When using cp --preserve=context to copy to an existing destination,
       reset the context as per the default context, which has already been
       set according to the src.
       When using the mutually exclusive -Z option, then adjust the type of
       the existing context according to the system default for the dest.
       Note we set the context here, _after_ the file is opened, lest the
       new context disallow that. */
    if (0 <= dest_desc
        && (x->set_security_context || x->preserve_security_context))
    {
        if (!set_file_security_ctx (dst_name, false, x))
        {
            if (x->require_preserve_context)
            {
                return_val = false;
                goto close_src_and_dst_desc;
            }
        }
    }

    if (dest_desc < 0 && dest_errno != ENOENT
        && x->unlink_dest_after_failed_open)
    {
        if (unlinkat (dst_dirfd, dst_relnname, 0) == 0)
        {
            if (x->verbose)
                printf (_("removed %s\n"), quoteaf (dst_name));
        }
        else if (errno != ENOENT)
        {
            error (0, errno, _("cannot remove %s"), quoteaf (dst_name));
        }
    }
}

```

```

        return_val = false;
        goto close_src_desc;
    }

    dest_errno = ENOENT;
}

if (dest_desc < 0 && dest_errno == ENOENT)
{
    /* Ensure there is no race where a file may be left without
       an appropriate security context. */
    if (x->set_security_context)
    {
        if (! set_process_security_ctx (src_name, dst_name, dst_mode,
                                       true, x))
        {
            return_val = false;
            goto close_src_desc;
        }
    }

    /* Tell caller that the destination file is created. */
    *new_dst = true;
}
}

if (*new_dst)
{
#ifndef HAVE_FCLONEFILEAT && !USE_XATTR
#ifndef CLONE_ACL
#define CLONE_ACL 0 /* Added in macOS 12.6. */
#endif
#endif
#ifndef CLONE_NOOWNERCOPY
#define CLONE_NOOWNERCOPY 0 /* Added in macOS 10.13. */
#endif
#endif
/* Try fclonefileat if copying data in reflink mode.
   Use CLONE_NOFOLLOW to avoid security issues that could occur
   if writing through dangling symlinks. Although the circa
   2023 macOS documentation doesn't say so, CLONE_NOFOLLOW
   affects the destination file too. */
if (data_copy_required && x->reflink_mode
    && (CLONE_NOOWNERCOPY || x->preserve_ownership))
{
    /* Try fclonefileat so long as it won't create the
       destination with unwanted permissions, which could lead
       to a security race. */
    mode_t cloned_mode_bits = S_ISVTX | S_IRWXUGO;
    mode_t cloned_mode = src_mode & cloned_mode_bits;
    mode_t desired_mode
        = (x->preserve_mode ? src_mode & CHMOD_MODE_BITS
          : x->set_mode ? x->mode
          : ((x->explicit_no_preserve_mode ? MODE_RW_UGO : dst_mode)
             & ~ cached_umask ()));
    if (!(cloned_mode & ~desired_mode))
    {
        int fc_flags
            = (CLONE_NOFOLLOW
              | (x->preserve_mode ? CLONE_ACL : 0)
              | (x->preserve_ownership ? 0 : CLONE_NOOWNERCOPY));
        int s = fclonefileat (source_desc, dst_dirfd, dst_relname,

```

```

                fc_flags);
if (s != 0 && (fc_flags & CLONE_ACL) && errno == EINVAL)
{
    fc_flags &= ~CLONE_ACL;
    s = fclonefileat (source_desc, dst_dirfd, dst_relname,
                      fc_flags);
}
if (s == 0)
{
    copy_debug.reflink = COPY_DEBUG_YES;

/* Update the clone's timestamps and permissions
   as needed. */

if (!x->preserve_timestamps)
{
    struct timespec timespec[2];
    timespec[0].tv_nsec = timespec[1].tv_nsec = UTIME_NOW;
    if (utimensat (dst_dirfd, dst_relname, timespec,
                   AT_SYMLINK_NOFOLLOW)
        != 0)
    {
        error (0, errno, _("updating times for %s"),
               quoteaf (dst_name));
        return_val = false;
        goto close_src_desc;
    }
}

extra_permissions = desired_mode & ~cloned_mode;
if (!extra_permissions
    && (!x->preserve_mode || (fc_flags & CLONE_ACL)
         || !fd_has_acl (source_desc)))
{
    goto close_src_desc;
}

/* Either some desired permissions were not cloned,
   or ACLs were not cloned despite that being requested. */
omitted_permissions = 0;
dest_desc = -1;
goto set_dest_mode;
}
if (! handle_clone_fail (dst_dirfd, dst_relname, src_name,
                        dst_name,
                        -1, false /* We didn't create dst */,
                        x->reflink_mode))
{
    return_val = false;
    goto close_src_desc;
}
else
    copy_debug.reflink = COPY_DEBUG_AVOIDED;
}
else if (data_copy_required && x->reflink_mode)
{
    if (! CLONE_NOOWNERCOPY)
        copy_debug.reflink = COPY_DEBUG_AVOIDED;
}

```

```

#endif

/* To allow copying xattrs on read-only files, create with u+w.
   This satisfies an inode permission check done by
   xattr_permission in fs/xattr.c of the GNU/Linux kernel. */
mode_t open_mode =
  ((dst_mode & ~omitted_permissions)
   | (preserve_xattr && !x->owner_privileges ? S_IWUSR : 0));
extra_permissions = open_mode & ~dst_mode; /* either 0 or S_IWUSR */

int open_flags = O_WRONLY | O_CREAT | O_BINARY;
dest_desc = openat (dst_dirfd, dst_relname, open_flags | O_EXCL,
                    open_mode);
dest_errno = errno;

/* When trying to copy through a dangling destination symlink,
   the above open fails with EEXIST. If that happens, and
   readlinkat shows that it is a symlink, then we
   have a problem: trying to resolve this dangling symlink to
   a directory/destination-entry pair is fundamentally racy,
   so punt. If x->open_dangling_dest_symlink is set (cp sets
   that when POSIXLY_CORRECT is set in the environment), simply
   call open again, but without O_EXCL (potentially dangerous).
   If not, fail with a diagnostic. These shenanigans are necessary
   only when copying, i.e., not in move_mode. */
if (dest_desc < 0 && dest_errno == EEXIST && !x->move_mode)
{
    {
    char dummy[1];
    if (0 <= readlinkat (dst_dirfd, dst_relname, dummy, sizeof dummy))
        {
        if (x->open_dangling_dest_symlink)
            {
            dest_desc = openat (dst_dirfd, dst_relname,
                                open_flags, open_mode);
            dest_errno = errno;
            }
        else
            {
            error (0, 0, _("not writing through dangling symlink %s"),
                   quoteaf (dst_name));
            return_val = false;
            goto close_src_desc;
            }
        }
    }
}

/* Improve quality of diagnostic when a nonexistent dst_name
   ends in a slash and open fails with errno == EISDIR. */
if (dest_desc < 0 && dest_errno == EISDIR
    && *dst_name && dst_name[strlen (dst_name) - 1] == '/')
    dest_errno = ENOTDIR;
}

else
{
    omitted_permissions = extra_permissions = 0;
}

if (dest_desc < 0)
{
    error (0, dest_errno, _("cannot create regular file %s"),

```

```

        quoteaf (dst_name));
return_val = false;
goto close_src_desc;
}

/* --attributes-only overrides --reflink. */
if (data_copy_required && x->reflink_mode)
{
    if (clone_file (dest_desc, source_desc) == 0)
    {
        data_copy_required = false;
        copy_debug.reflink = COPY_DEBUG_YES;
    }
else
{
    if (! handle_clone_fail (dst_dirfd, dst_relname, src_name, dst_name,
                           dest_desc, *new_dst, x->reflink_mode))
    {
        return_val = false;
        goto close_src_and_dst_desc;
    }
}
}

if (! (data_copy_required | x->preserve_ownership | extra_permissions))
    sb.st_mode = 0;
else if (fstat (dest_desc, &sb) != 0)
{
    error (0, errno, _("cannot fstat %s"), quoteaf (dst_name));
    return_val = false;
    goto close_src_and_dst_desc;
}

/* If extra permissions needed for copy_xattr didn't happen (e.g.,
   due to umask) chmod to add them temporarily; if that fails give
   up with extra permissions, letting copy_attr fail later. */
mode_t temporary_mode = sb.st_mode | extra_permissions;
if (temporary_mode != sb.st_mode
    && (fchmod_or_lchmod (dest_desc, dst_dirfd, dst_relname, temporary_mode)
         != 0))
    extra_permissions = 0;

if (data_copy_required)
{
    /* Choose a suitable buffer size; it may be adjusted later. */
    size_t buf_size = io_blksize (&sb);
    size_t hole_size = STP_BLKSIZE (&sb);

    /* Deal with sparse files. */
    enum scantype scantype = infer_scantype (source_desc, &src_open_sb,
                                              &scan_inference);
    if (scantype == ERROR_SCANTYPE)
    {
        error (0, errno, _("cannot lseek %s"), quoteaf (src_name));
        return_val = false;
        goto close_src_and_dst_desc;
    }
    bool make_holes
        = (S_ISREG (sb.st_mode)
           && (x->sparse_mode == SPARSE_ALWAYS

```

```

|| (x->sparse_mode == SPARSE_AUTO
    && scantype != PLAIN_SCANTYPE));

fdadvise (source_desc, 0, 0, FADVISE_SEQUENTIAL);

/* If not making a sparse file, try to use a more-efficient
   buffer size. */
if (! make_holes)
{
    /* Compute the least common multiple of the input and output
       buffer sizes, adjusting for outlandish values.
       Note we read in multiples of the reported block size
       to support (unusual) devices that have this constraint. */
    size_t blcm_max = MIN (SIZE_MAX, SSIZE_MAX);
    size_t blcm = buffer_lcm (io_blksize (&src_open_sb), buf_size,
                             blcm_max);

    /* Do not bother with a buffer larger than the input file, plus one
       byte to make sure the file has not grown while reading it. */
    if (S_ISREG (src_open_sb.st_mode) && src_open_sb.st_size < buf_size)
        buf_size = src_open_sb.st_size + 1;

    /* However, stick with a block size that is a positive multiple of
       blcm, overriding the above adjustments. Watch out for
       overflow. */
    buf_size += blcm - 1;
    buf_size -= buf_size % blcm;
    if (buf_size == 0 || blcm_max < buf_size)
        buf_size = blcm;
}

off_t n_read;
bool wrote_hole_at_eof = false;
if (! (
#define SEEK_HOLE
    scantype == LSEEK_SCANTYPE
    ? lseek_copy (source_desc, dest_desc, &buf, buf_size, hole_size,
                  scan_inference.ext_start, src_open_sb.st_size,
                  make_holes ? x->sparse_mode : SPARSE_NEVER,
                  x->reflink_mode != REFLINK_NEVER,
                  src_name, dst_name)
    :
#endif
    sparse_copy (source_desc, dest_desc, &buf, buf_size,
                 make_holes ? hole_size : 0,
                 x->sparse_mode == SPARSE_ALWAYS,
                 x->reflink_mode != REFLINK_NEVER,
                 src_name, dst_name, UINTMAX_MAX, &n_read,
                 &wrote_hole_at_eof))
{
    return_val = false;
    goto close_src_and_dst_desc;
}
else if (wrote_hole_at_eof && ftruncate (dest_desc, n_read) < 0)
{
    error (0, errno, _("failed to extend %s"), quoteaf (dst_name));
    return_val = false;
    goto close_src_and_dst_desc;
}
}

```

```

if (x->preserve_timestamps)
{
    struct timespec timespec[2];
    timespec[0] = get_stat_atime (src_sb);
    timespec[1] = get_stat_mtime (src_sb);

    if (fdutimensat (dest_desc, dst_dirfd, dst_relname, timespec, 0) != 0)
    {
        error (0, errno, _("preserving times for %s"), quoteaf (dst_name));
        if (x->require_preserve)
        {
            return_val = false;
            goto close_src_and_dst_desc;
        }
    }
}

/* Set ownership before xattrs as changing owners will
   clear capabilities. */
if (x->preserve_ownership && ! SAME_OWNER_AND_GROUP (*src_sb, sb))
{
    switch (set_owner (x, dst_name, dst_dirfd, dst_relname, dest_desc,
                       src_sb, *new_dst, &sb))
    {
        case -1:
            return_val = false;
            goto close_src_and_dst_desc;

        case 0:
            src_mode &= ~ (S_ISUID | S_ISGID | S_ISVTX);
            break;
    }
}

if (preserve_xattr)
{
    if (!copy_attr (src_name, source_desc, dst_name, dest_desc, x)
        && x->require_preserve_xattr)
        return_val = false;
}

set_author (dst_name, dest_desc, src_sb);

#if HAVE_FCLONEFILEAT && !USE_XATTR
set_dest_mode:
#endif
if (x->preserve_mode || x->move_mode)
{
    if (copy_acl (src_name, source_desc, dst_name, dest_desc, src_mode) != 0
        && x->require_preserve)
        return_val = false;
}
else if (x->set_mode)
{
    if (set_acl (dst_name, dest_desc, x->mode) != 0)
        return_val = false;
}
else if (x->explicit_no_preserve_mode && *new_dst)
{

```

```

        if (set_acl (dst_name, dest_desc, MODE_RW_UGO & ~cached_umask ()) != 0)
            return_val = false;
    }
else if (omitted_permissions | extra_permissions)
{
    omitted_permissions &= ~ cached_umask ();
    if ((omitted_permissions | extra_permissions)
        && (fchmod_or_lchmod (dest_desc, dst_dirfd, dst_relname,
                             dst_mode & ~ cached_umask ())
            != 0))
    {
        error (0, errno, _("preserving permissions for %s"),
               quoteaf (dst_name));
        if (x->require_preserve)
            return_val = false;
    }
}

if (dest_desc < 0)
    goto close_src_desc;

close_src_and_dst_desc:
if (close (dest_desc) < 0)
{
    error (0, errno, _("failed to close %s"), quoteaf (dst_name));
    return_val = false;
}
close_src_desc:
if (close (source_desc) < 0)
{
    error (0, errno, _("failed to close %s"), quoteaf (src_name));
    return_val = false;
}

/* Output debug info for data copying operations. */
if (x->debug)
    emit_debug (x);

alignfree (buf);
return return_val;
}

/* Return whether it's OK that two files are the "same" by some measure.
The first file is SRC_NAME and has status SRC_SB.
The second is DST_DIRFD+DST_RELNAME and has status DST_SB.
The copying options are X. The goal is to avoid
making the 'copy' operation remove both copies of the file
in that case, while still allowing the user to e.g., move or
copy a regular file onto a symlink that points to it.
Try to minimize the cost of this function in the common case.
Set *RETURN_NOW if we've determined that the caller has no more
work to do and should return successfully, right away. */

static bool
same_file_ok (char const *src_name, struct stat const *src_sb,
              int dst_dirfd, char const *dst_relname, struct stat const *dst_sb,
              const struct cp_options *x, bool *return_now)
{
const struct stat *src_sb_link;
const struct stat *dst_sb_link;

```

```

struct stat tmp_dst_sb;
struct stat tmp_src_sb;

bool same_link;
bool same = psame_inode (src_sb, dst_sb);

*return_now = false;

/* FIXME: this should (at the very least) be moved into the following
   if-block. More likely, it should be removed, because it inhibits
   making backups. But removing it will result in a change in behavior
   that will probably have to be documented -- and tests will have to
   be updated. */
if (same && x->hard_link)
{
    *return_now = true;
    return true;
}

if (x->dereference == DEREF_NEVER)
{
    same_link = same;

    /* If both the source and destination files are symlinks (and we'll
       know this here IFF preserving symlinks), then it's usually ok
       when they are distinct. */
    if (S_ISLNK (src_sb->st_mode) && S_ISLNK (dst_sb->st_mode))
    {
        bool sn = same_nameat (AT_FDCWD, src_name, dst_dirfd, dst_relname);
        if (!sn)
        {
            /* It's fine when we're making any type of backup. */
            if (x->backup_type != no_backups)
                return true;

            /* Here we have two symlinks that are hard-linked together,
               and we're not making backups. In this unusual case, simply
               returning true would lead to mv calling "rename(A,B)",
               which would do nothing and return 0. */
            if (same_link)
            {
                *return_now = true;
                return !x->move_mode;
            }
        }
    }

    return !sn;
}

src_sb_link = src_sb;
dst_sb_link = dst_sb;
}
else
{
    if (!same)
        return true;

    if (fstatat (dst_dirfd, dst_relname, &tmp_dst_sb,
                  AT_SYMLINK_NOFOLLOW) != 0
        || lstat (src_name, &tmp_src_sb) != 0)

```

```

        return true;

src_sb_link = &tmp_src_sb;
dst_sb_link = &tmp_dst_sb;

same_link = psame_inode (src_sb_link, dst_sb_link);

/* If both are symlinks, then it's ok, but only if the destination
   will be unlinked before being opened. This is like the test
   above, but with the addition of the unlink_dest_before_opening
   conjunct because otherwise, with two symlinks to the same target,
   we'd end up truncating the source file. */
if (S_ISLNK (src_sb_link->st_mode) && S_ISLNK (dst_sb_link->st_mode)
    && x->unlink_dest_before_opening)
    return true;
}

/* The backup code ensures there's a copy, so it's usually ok to
   remove any destination file. One exception is when both
   source and destination are the same directory entry. In that
   case, moving the destination file aside (in making the backup)
   would also rename the source file and result in an error. */
if (x->backup_type != no_backups)
{
    if (!same_link)
    {
        /* In copy mode when dereferencing symlinks, if the source is a
           symlink and the dest is not, then backing up the destination
           (moving it aside) would make it a dangling symlink, and the
           subsequent attempt to open it in copy_reg would fail with
           a misleading diagnostic. Avoid that by returning zero in
           that case so the caller can make cp (or mv when it has to
           resort to reading the source file) fail now. */

        /* FIXME-note: even with the following kludge, we can still provoke
           the offending diagnostic. It's just a little harder to do :-)
           $ rm -f a b c; touch c; ln -s c b; ln -s b a; cp -b a b
           cp: cannot open 'a' for reading: No such file or directory
           That's misleading, since a subsequent 'ls' shows that 'a'
           is still there.
           One solution would be to open the source file *before* moving
           aside the destination, but that'd involve a big rewrite. */
        if ( ! x->move_mode
            && x->dereference != DEREF_NEVER
            && S_ISLNK (src_sb_link->st_mode)
            && ! S_ISLNK (dst_sb_link->st_mode))
            return false;

        return true;
    }

    /* FIXME: What about case insensitive file systems ? */
    return ! same_nameat (AT_FDCWD, src_name, dst_dirfd, dst_relname);
}

#endif
/* FIXME: use or remove */

/* If we're making a backup, we'll detect the problem case in
   copy_reg because SRC_NAME will no longer exist. Allowing

```

the test to be deferred lets cp do some useful things.  
 But when creating hardlinks and SRC\_NAME is a symlink  
 but DST\_RELNAME is not we must test anyway. \*/

```

if (x->hard_link
    || !S_ISLNK (src_sb_link->st_mode)
    || S_ISLNK (dst_sb_link->st_mode))
    return true;

if (x->dereference != DEREF_NEVER)
    return true;
#endif

if (x->move_mode || x->unlink_dest_before_opening)
{
    /* They may refer to the same file if we're in move mode and the
       target is a symlink. That is ok, since we remove any existing
       destination file before opening it -- via 'rename' if they're on
       the same file system, via unlinkat otherwise. */
    if (S_ISLNK (dst_sb_link->st_mode))
        return true;

    /* It's not ok if they're distinct hard links to the same file as
       this causes a race condition and we may lose data in this case. */
    if (same_link
        && 1 < dst_sb_link->st_nlink
        && ! same_nameat (AT_FDCWD, src_name, dst_dirfd, dst_relnname))
        return ! x->move_mode;
}

/* If neither is a symlink, then it's ok as long as they aren't
   hard links to the same file. */
if (!S_ISLNK (src_sb_link->st_mode) && !S_ISLNK (dst_sb_link->st_mode))
{
    if (!psame_inode (src_sb_link, dst_sb_link))
        return true;

    /* If they are the same file, it's ok if we're making hard links. */
    if (x->hard_link)
    {
        *return_now = true;
        return true;
    }
}

/* At this point, it is normally an error (data loss) to move a symlink
   onto its referent, but in at least one narrow case, it is not:
   In move mode, when
   1) src is a symlink,
   2) dest has a link count of 2 or more and
   3) dest and the referent of src are not the same directory entry,
      then it's ok, since while we'll lose one of those hard links,
      src will still point to a remaining link.
   Note that technically, condition #3 obviates condition #2, but we
   retain the 1 < st_nlink condition because that means fewer invocations
   of the more expensive #3.
```

Given this,

```
$ touch f && ln f l && ln -s f s
$ ls -og f l s
-rw-----. 2 0 Jan 4 22:46 f
```

```

-rw----- 2 0 Jan 4 22:46 l
lrwxrwxrwx. 1 1 Jan 4 22:46 s -> f
this must fail: mv s f
this must succeed: mv s l */

if (x->move_mode
    && S_ISLNK (src_sb->st_mode)
    && 1 < dst_sb_link->st_nlink)
{
    char *abs_src = canonicalize_file_name (src_name);
    if (abs_src)
    {
        bool result = ! same_nameat (AT_FDCWD, abs_src,
                                      dst_dirfd, dst_relname);
        free (abs_src);
        return result;
    }
}

/* It's ok to recreate a destination symlink. */
if (x->symbolic_link && S_ISLNK (dst_sb_link->st_mode))
    return true;

if (x->dereference == DEREF_NEVER)
{
    if (! S_ISLNK (src_sb_link->st_mode))
        tmp_src_sb = *src_sb_link;
    else if (stat (src_name, &tmp_src_sb) != 0)
        return true;

    if (! S_ISLNK (dst_sb_link->st_mode))
        tmp_dst_sb = *dst_sb_link;
    else if (fstatat (dst_dirfd, dst_relname, &tmp_dst_sb, 0) != 0)
        return true;

    if (!psame_inode (&tmp_src_sb, &tmp_dst_sb))
        return true;

    if (x->hard_link)
    {
        /* It's ok to attempt to hardlink the same file,
           and return early if not replacing a symlink.
           Note we need to return early to avoid a later
           unlink() of DST (when SRC is a symlink). */
        *return_now = ! S_ISLNK (dst_sb_link->st_mode);
        return true;
    }
}

return false;
}

/* Return whether DST_DIRFD+DST_RELNAME, with mode MODE,
   is writable in the sense of 'mv'.
   Always consider a symbolic link to be writable. */
static bool
writable_destination (int dst_dirfd, char const *dst_relname, mode_t mode)
{
    return (S_ISLNK (mode)
            || can_write_any_file ()
            || faccessat (dst_dirfd, dst_relname, W_OK, AT_EACCESS) == 0);
}

```

```

}

static bool
overwrite_ok (struct cp_options const *x, char const *dst_name,
              int dst_dirfd, char const *dst_relname,
              struct stat const *dst_sb)
{
if (! writable_destination (dst_dirfd, dst_relname, dst_sb->st_mode))
{
    {
char perms[12];           /* "-rwxrwxrwx " ls-style modes. */
strmode (dst_sb->st_mode, perms);
perms[10] = '\0';
fprintf (stderr,
        (x->move_mode || x->unlink_dest_before_opening
         || x->unlink_dest_after_failed_open)
        ? _(""%s: replace %s, overriding mode %04lo (%s)? ")
        : _(""%s: unwritable %s (mode %04lo, %s); try anyway? "),
        program_name, quoteaf (dst_name),
        (unsigned long int) (dst_sb->st_mode & CHMOD_MODE_BITS),
        &perms[1]);
    }
}
else
{
    {
fprintf (stderr, _(""%s: overwrite %s? "),
        program_name, quoteaf (dst_name));
    }
}

return yesno ();
}

/* Initialize the hash table implementing a set of F_triple entries
corresponding to destination files. */
extern void
dest_info_init (struct cp_options *x)
{
x->dest_info
    = hash_initialize (DEST_INFO_INITIAL_CAPACITY,
                      nullptr,
                      triple_hash,
                      triple_compare,
                      triple_free);
if (! x->dest_info)
    xalloc_die ();
}

/* Initialize the hash table implementing a set of F_triple entries
corresponding to source files listed on the command line. */
extern void
src_info_init (struct cp_options *x)
{
/* Note that we use triple_hash_no_name here.
   Contrast with the use of triple_hash above.
   That is necessary because a source file may be specified
   in many different ways. We want to warn about this
   cp a a d/
   as well as this:
   cp a ./a d/
*/
x->src_info

```

```

= hash_initialize (DEST_INFO_INITIAL_CAPACITY,
                  nullptr,
                  triple_hash_no_name,
                  triple_compare,
                  triple_free);
if (!x->src_info)
    xalloc_die ();
}

/* When effecting a move (e.g., for mv(1)), and given the name DST_NAME
aka DST_DIRFD+DST_RELNAME
of the destination and a corresponding stat buffer, DST_SB, return
true if the logical 'move' operation should _not_ proceed.
Otherwise, return false.
Depending on options specified in X, this code may issue an
interactive prompt asking whether it's ok to overwrite DST_NAME. */
static bool
abandon_move (const struct cp_options *x,
              char const *dst_name,
              int dst_dirfd, char const *dst_relname,
              struct stat const *dst_sb)
{
affirm (x->move_mode);
return (x->interactive == I_ALWAYS_NO
        || x->interactive == I_ALWAYS_SKIP
        || ((x->interactive == I_ASK_USER
              || (x->interactive == I_UNSPECIFIED
                  && x->stdin_tty
                  && !writable_destination (dst_dirfd, dst_relname,
                                             dst_sb->st_mode)))
            && !overwrite_ok (x, dst_name, dst_dirfd, dst_relname, dst_sb)));
}

/* Print --verbose output on standard output, e.g. 'new' -> 'old'.
If BACKUP_DST_NAME is non-null, then also indicate that it is
the name of a backup file. */
static void
emit_verbose (char const *format, char const *src, char const *dst,
              char const *backup_dst_name)
{
printf (format, quoteaf_n (0, src), quoteaf_n (1, dst));
if (backup_dst_name)
    printf (_(" (backup: %s)"), quoteaf (backup_dst_name));
putchar ('\n');
}

/* A wrapper around "setfscreatecon (nullptr)" that exits upon failure. */
static void
restore_default_fscreatecon_or_die (void)
{
if (setfscreatecon (nullptr) != 0)
    error (EXIT_FAILURE, errno,
          _("failed to restore the default file creation context"));
}

/* Return a newly-allocated string that is like STR
except replace its suffix SUFFIX with NEWSUFFIX. */
static char *
subst_suffix (char const *str, char const *suffix, char const *newsuffix)
{

```

```

idx_t prefixlen = suffix - str;
idx_t newsuffixsize = strlen (newsuffix) + 1;
char *r = ximalloc (prefixlen + newsuffixsize);
memcpy (r + prefixlen, newsuffix, newsuffixsize);
return memcpy (r, str, prefixlen);
}

/* Create a hard link to SRC_NAME aka SRC_DIRFD+SRC_RELNAME;
the new link is at DST_NAME aka DST_DIRFD+DST_RELNAME.
A null SRC_NAME stands for the file whose name is like DST_NAME
except with DST_RELNAME replaced with SRC_RELNAME.
Honor the REPLACE, VERBOSE and DEREFERENCE settings.
Return true upon success. Otherwise, diagnose the
failure and return false. If SRC_NAME is a symbolic link, then it will not
be followed unless DEREFERENCE is true.
If the system doesn't support hard links to symbolic links, then DST_NAME
will be created as a symbolic link to SRC_NAME. */
static bool
create_hard_link (char const *src_name, int src_dirfd, char const *src_relname,
                  char const *dst_name, int dst_dirfd, char const *dst_relname,
                  bool replace, bool verbose, bool dereference)
{
int err = force_linkat (src_dirfd, src_relname, dst_dirfd, dst_relname,
                        dereference ? AT_SYMLINK_FOLLOW : 0,
                        replace, -1);
if (0 < err)
{
    char *a_src_name = nullptr;
    if (!src_name)
        src_name = a_src_name = subst_suffix (dst_name, dst_relname,
                                              src_relname);
    error (0, err, _("cannot create hard link %s to %s"),
           quoteaf_n (0, dst_name), quoteaf_n (1, src_name));
    free (a_src_name);
    return false;
}
if (err < 0 && verbose)
    printf (_("removed %s\n"), quoteaf (dst_name));
return true;
}

/* Return true if the current file should be (tried to be) dereferenced:
either for DEREF_ALWAYS or for DEREF_COMMAND_LINE_ARGUMENTS in the case
where the current file is a COMMAND_LINE_ARG; otherwise return false. */
ATTRIBUTE_PURE
static inline bool
should_dereference (const struct cp_options *x, bool command_line_arg)
{
return x->dereference == DEREF_ALWAYS
    || (x->dereference == DEREF_COMMAND_LINE_ARGUMENTS
        && command_line_arg);
}

/* Return true if the source file with basename SRCBASE and status SRC_ST
is likely to be the simple backup file for DST_DIRFD+DST_RELNAME. */
static bool
source_is_dst_backup (char const *srcbase, struct stat const *src_st,
                      int dst_dirfd, char const *dst_relname)
{

```

```

size_t srcbaselen = strlen (srcbase);
char const *dstbase = last_component (dst_relname);
size_t dstbaselen = strlen (dstbase);
size_t suffixlen = strlen (simple_backup_suffix);
if (! (srcbaselen == dstbaselen + suffixlen
      && memcmp (srcbase, dstbase, dstbaselen) == 0
      && STREQ (srcbase + dstbaselen, simple_backup_suffix)))
    return false;
char *dst_back = subst_suffix (dst_relname,
                               dst_relname + strlen (dst_relname),
                               simple_backup_suffix);
struct stat dst_back_sb;
int dst_back_status = fstatat (dst_dirfd, dst_back, &dst_back_sb, 0);
free (dst_back);
return dst_back_status == 0 && psame_inode (src_st, &dst_back_sb);
}

/* Copy the file SRC_NAME to the file DST_NAME aka DST_DIRFD+DST_RELNAME.
If NONEXISTENT_DST is positive, DST_NAME does not exist even as a
dangling symlink; if negative, it does not exist except possibly
as a dangling symlink; if zero, its existence status is unknown.
A non-null PARENT describes the parent directory.
ANCESTORS points to a linked, null terminated list of
devices and inodes of parent directories of SRC_NAME.
X summarizes the command-line options.
COMMAND_LINE_ARG means SRC_NAME was specified on the command line.
FIRST_DIR_CREATED_PER_COMMAND_LINE_ARG is both input and output.
Set *COPY_INTO_SELF if SRC_NAME is a parent of (or the
same as) DST_NAME; otherwise, clear it.
If X->move_mode, set *RENAME_SUCCEEDED according to whether
the source was simply renamed to the destination.
Return true if successful. */
static bool
copy_internal (char const *src_name, char const *dst_name,
               int dst_dirfd, char const *dst_relname,
               int nonexistent_dst,
               struct stat const *parent,
               struct dir_list *ancestors,
               const struct cp_options *x,
               bool command_line_arg,
               bool *first_dir_created_per_command_line_arg,
               bool *copy_into_self,
               bool *rename_succeeded)
{
struct stat src_sb;
struct stat dst_sb;
mode_t src_mode IF_LINT (= 0);
mode_t dst_mode IF_LINT (= 0);
mode_t dst_mode_bits;
mode_t omitted_permissions;
bool restore_dst_mode = false;
char *earlier_file = nullptr;
char *dst_backup = nullptr;
char const *drelname = *dst_relname ? dst_relname : ".";
bool delayed_ok;
bool copied_as_regular = false;
bool dest_is_symlink = false;
bool have_dst_lstat = false;

*copy_into_self = false;

```

```

int rename_errno = x->rename_errno;
if (x->move_mode && !x->exchange)
{
    if (rename_errno < 0)
        rename_errno = (renameatu (AT_FDCWD, src_name, dst_dirfd, drelname,
                                   RENAME_NOREPLACE)
                        ? errno : 0);
    nonexistent_dst = *rename_succeeded = rename_errno == 0;
}

if (rename_errno == 0
    ? !x->last_file
    : rename_errno != EEXIST
    || (x->interactive != I_ALWAYS_NO && x->interactive != I_ALWAYS_SKIP))
{
    char const *name = rename_errno == 0 ? dst_name : src_name;
    int dirfd = rename_errno == 0 ? dst_dirfd : AT_FDCWD;
    char const *relname = rename_errno == 0 ? drelname : src_name;
    int fstatat_flags
        = x->dereference == DEREF_NEVER ? AT_SYMLINK_NOFOLLOW : 0;
    if (follow_fstatat (dirfd, relname, &src_sb, fstatat_flags) != 0)
    {
        error (0, errno, _("cannot stat %s"), quoteaf (name));
        return false;
    }
}

src_mode = src_sb.st_mode;

if (S_ISDIR (src_mode) && !x->recursive)
{
    error (0, 0, !x->install_mode /* cp */
           ? _("-r not specified; omitting directory %s")
           : _("omitting directory %s"),
           quoteaf (src_name));
    return false;
}
else
{
    #if defined lint && (defined __clang__ || defined __COVERITY__)
        affirm (x->move_mode);
        memset (&src_sb, 0, sizeof src_sb);
    #endif
}

/* Detect the case in which the same source file appears more than
   once on the command line and no backup option has been selected.
   If so, simply warn and don't copy it the second time.
   This check is enabled only if x->src_info is non-null. */
if (command_line_arg && x->src_info)
{
    if (!S_ISDIR (src_mode)
        && x->backup_type == no_backups
        && seen_file (x->src_info, src_name, &src_sb))
    {
        error (0, 0, _("warning: source file %s specified more than once"),
               quoteaf (src_name));
        return true;
    }
}

```

```

record_file (x->src_info, src_name, &src_sb);
}

bool dereference = should_dereference (x, command_line_arg);

/* Whether the destination is (or was) known to be new, updated as
   more info comes in. This may become true if the destination is a
   dangling symlink, in contexts where dangling symlinks should be
   treated the same as nonexistent files. */
bool new_dst = 0 < nonexistent_dst;

if (! new_dst)
{
    /* Normally, fill in DST_SB or set NEW_DST so that later code
       can use DST_SB if NEW_DST is false. However, don't bother
       doing this when rename_errno == EEXIST and X->interactive is
       I_ALWAYS_NO or I_ALWAYS_SKIP, something that can happen only
       with mv in which case x->update must be false which means
       that even if !NEW_DST the move will be abandoned without
       looking at DST_SB. */
    if (! (rename_errno == EEXIST
        && (x->interactive == I_ALWAYS_NO
            || x->interactive == I_ALWAYS_SKIP)))
    {
        /* Regular files can be created by writing through symbolic
           links, but other files cannot. So use stat on the
           destination when copying a regular file, and lstat otherwise.
           However, if we intend to unlink or remove the destination
           first, use lstat, since a copy won't actually be made to the
           destination in that case. */
        bool use_lstat
            = (! S_ISREG (src_mode)
                && (! x->copy_as_regular
                    || (S_ISDIR (src_mode) && !x->keep_directory_symlink)
                    || S_ISLNK (src_mode)))
                || x->move_mode || x->symbolic_link || x->hard_link
                || x->backup_type != no_backups
                || x->unlink_dest_before_opening);
        if (!use_lstat && nonexistent_dst < 0)
            new_dst = true;
        else if (0 <= follow_fstatat (dst_dirfd, drelname, &dst_sb,
            use_lstat ? AT_SYMLINK_NOFOLLOW : 0))
        {
            have_dst_lstat = use_lstat;
            rename_errno = EEXIST;
        }
        else if (errno == ENOENT)
            new_dst = true;
        else if (errno == ELOOP && !use_lstat
            && x->unlink_dest_after_failed_open)
        {
            /* cp -f's destination might be a symlink loop.
               Leave new_dst=false so that we try to unlink later. */
        }
        else
        {
            error (0, errno, _("cannot stat %s"), quoteaf (dst_name));
            return false;
        }
    }
}

```

```

    }

if (rename_errno == EEXIST)
{
    bool return_now = false;
    bool return_val = true;
    bool skipped = false;

    if ((x->interactive != I_ALWAYS_NO && x->interactive != I_ALWAYS_SKIP)
        && ! same_file_ok (src_name, &src_sb, dst_dirfd, drelname,
                            &dst_sb, x, &return_now))
    {
        error (0, 0, _("'%s and %s are the same file"),
               quoteaf_n (0, src_name), quoteaf_n (1, dst_name));
        return false;
    }

if (x->update && !S_ISDIR (src_mode))
{
    /* When preserving timestamps (but not moving within a file
       system), don't worry if the destination timestamp is
       less than the source merely because of timestamp
       truncation. */
    int options = ((x->preserve_timestamps
                    && ! (x->move_mode
                           && dst_sb.st_dev == src_sb.st_dev))
                  ? UTIMECMP_TRUNCATE_SOURCE
                  : 0);

    if (0 <= utimecmpat (dst_dirfd, dst_relname, &dst_sb,
                          &src_sb, options))
    {
        /* We're using --update and the destination is not older
           than the source, so do not copy or move. Pretend the
           rename succeeded, so the caller (if it's mv) doesn't
           end up removing the source file. */
        if (rename_succeeded)
            *rename_succeeded = true;

        /* However, we still must record that we've processed
           this src/dest pair, in case this source file is
           hard-linked to another one. In that case, we'll use
           the mapping information to link the corresponding
           destination names. */
        earlier_file = remember_copied (dst_relname, src_sb.st_ino,
                                         src_sb.st_dev);
        if (earlier_file)
        {
            /* Note we currently replace DST_NAME unconditionally,
               even if it was a newer separate file. */
            if (! create_hard_link (nullptr, dst_dirfd, earlier_file,
                                   dst_name, dst_dirfd, dst_relname,
                                   true,
                                   x->verbose, dereference))
            {
                goto un_backup;
            }
        }
    }

skipped = true;
}

```

```

        goto skip;
    }
}

/* When there is an existing destination file, we may end up
   returning early, and hence not copying/moving the file.
   This may be due to an interactive 'negative' reply to the
   prompt about the existing file. It may also be due to the
   use of the --no-clobber option.

cp and mv treat -i and -f differently. */
if (x->move_mode)
{
    if (abandon_move (x, dst_name, dst_dirfd, drelname, &dst_sb))
    {
        /* Pretend the rename succeeded, so the caller (mv)
           doesn't end up removing the source file. */
        if (rename_succeeded)
            *rename_succeeded = true;

        skipped = true;
        return_val = x->interactive == I_ALWAYS_SKIP;
    }
}
else
{
    if (! S_ISDIR (src_mode)
        && (x->interactive == I_ALWAYS_NO
            || x->interactive == I_ALWAYS_SKIP
            || (x->interactive == I_ASK_USER
                && ! overwrite_ok (x, dst_name, dst_dirfd,
                    dst_relname, &dst_sb))))
    {
        skipped = true;
        return_val = x->interactive == I_ALWAYS_SKIP;
    }
}

skip:
if (skipped)
{
    if (x->interactive == I_ALWAYS_NO)
        error (0, 0, _("not replacing %s"), quoteaf (dst_name));
    else if (x->debug)
        printf (_("skipped %s\n"), quoteaf (dst_name));

    return_now = true;
}

if (return_now)
    return return_val;

/* Copying a directory onto a non-directory, or vice versa,
   is ok only with --backup or --exchange. */
if (!S_ISDIR (src_mode) != !S_ISDIR (dst_sb.st_mode)
    && x->backup_type == no_backups && !x->exchange)
{
    error (0, 0,
        _("%s\n"
        ? ("cannot overwrite non-directory %s "

```

```

        "with directory %s")
    : ("cannot overwrite directory %s "
        "with non-directory %s"),
        quoteaf_n (0, dst_name), quoteaf_n (1, src_name));
return false;
}

/* Don't let the user destroy their data, even if they try hard:
   This mv command must fail (likewise for cp):
   rm -rf a b c; mkdir a b c; touch a/f b/f; mv a/f b/f c
   Otherwise, the contents of b/f would be lost.
   In the case of 'cp', b/f would be lost if the user simulated
   a move using cp and rm.
   Nothing is lost if you use --backup=numbered or --exchange. */
if (!S_ISDIR (dst_sb.st_mode) && command_line_arg
    && x->backup_type != numbered_backups && !x->exchange
    && seen_file (x->dest_info, dst_relname, &dst_sb))
{
error (0, 0,
      _("will not overwrite just-created %s with %s"),
      quoteaf_n (0, dst_name), quoteaf_n (1, src_name));
return false;
}

char const *srcbase;
if (x->backup_type != no_backups
    /* Don't try to back up a destination if the last
       component of src_name is "." or "..". */
    && ! dot_or_dotdot (srcbase = last_component (src_name)))
/* Create a backup of each destination directory in move mode,
   but not in copy mode. FIXME: it might make sense to add an
   option to suppress backup creation also for move mode.
   That would let one use mv to merge new content into an
   existing hierarchy. */
    && (x->move_mode || ! S_ISDIR (dst_sb.st_mode)))
{
/* Fail if creating the backup file would likely destroy
   the source file. Otherwise, the commands:
   cd /tmp; rm -f a a~; : > a; echo A > a~; cp --b=simple a~ a
   would leave two zero-length files: a and a~. */
if (x->backup_type != numbered_backups
    && source_is_dst_backup (srcbase, &src_sb,
                               dst_dirfd, dst_relname))
{
char const *fmt;
fmt = (x->move_mode
      ? _("Backing up %s might destroy source; %s not moved")
      : _("Backing up %s might destroy source; %s not copied"));
error (0, 0, fmt,
      quoteaf_n (0, dst_name),
      quoteaf_n (1, src_name));
return false;
}

char *tmp_backup = backup_file_rename (dst_dirfd, dst_relname,
                                       x->backup_type);

/* FIXME: use fts:
   Using alloca for a file name that may be arbitrarily
   long is not recommended. In fact, even forming such a name

```

```

should be discouraged. Eventually, this code will be rewritten
to use fts, so using alloca here will be less of a problem. */
if (tmp_backup)
{
    idx_t dirlen = dst_relname - dst_name;
    idx_t backupsize = strlen (tmp_backup) + 1;
    dst_backup = alloca (dirlen + backupsize);
    mempcpy (mempcpy (dst_backup, dst_name, dirlen),
              tmp_backup, backupsize);
    free (tmp_backup);
}
else if (errno != ENOENT)
{
    error (0, errno, _("cannot backup %s"), quoteaf (dst_name));
    return false;
}
new_dst = true;
}
else if (! S_ISDIR (dst_sb.st_mode)
/* Never unlink dst_name when in move mode. */
&& ! x->move_mode
&& (x->unlink_dest_before_opening
|| (x->data_copy_required
&& ((x->preserve_links && 1 < dst_sb.st_nlink)
|| (x->dereference == DEREF_NEVER
&& ! S_ISREG (src_sb.st_mode))))
))
{
if (unlinkat (dst_dirfd, dst_relname, 0) != 0 && errno != ENOENT)
{
    error (0, errno, _("cannot remove %s"), quoteaf (dst_name));
    return false;
}
new_dst = true;
if (x->verbose)
    printf (_("removed %s\n"), quoteaf (dst_name));
}
}
/* Ensure we don't try to copy through a symlink that was
created by a prior call to this function. */
if (command_line_arg
&& x->dest_info
&& ! x->move_mode
&& x->backup_type == no_backups)
{
/* If we did not follow symlinks above, good: use that data.
Otherwise, use AT_SYMLINK_NOFOLLOW, in case dst_name is a symlink. */
struct stat tmp_buf;
struct stat *dst_lstat_sb
= (have_dst_lstat ? &dst_sb
: fstatat (dst_dirfd, drename, &tmp_buf, AT_SYMLINK_NOFOLLOW) < 0
? nullptr : &tmp_buf);

/* Never copy through a symlink we've just created. */
if (dst_lstat_sb
&& S_ISLNK (dst_lstat_sb->st_mode)
&& seen_file (x->dest_info, dst_relname, dst_lstat_sb))
{

```

```

        error (0, 0,
            _("will not copy %s through just-created symlink %s"),
            quoteaf_n (0, src_name), quoteaf_n (1, dst_name));
    return false;
}
}

/* If the source is a directory, we don't always create the destination
   directory. So --verbose should not announce anything until we're
   sure we'll create a directory. Also don't announce yet when moving
   so we can distinguish renames versus copies. */
if (x->verbose && !x->move_mode && !S_ISDIR (src_mode))
    emit_verbose ("%s -> %s", src_name, dst_name, dst_backup);

/* Associate the destination file name with the source device and inode
   so that if we encounter a matching dev/ino pair in the source tree
   we can arrange to create a hard link between the corresponding names
   in the destination tree.

```

When using the --link (-l) option, there is no need to take special measures, because (barring race conditions) files that are hard-linked in the source tree will also be hard-linked in the destination tree.

Sometimes, when preserving links, we have to record dev/ino even though st\_nlink == 1:

- when in move\_mode, since we may be moving a group of N hard-linked files (via two or more command line arguments) to a different partition; the links may be distributed among the command line arguments (possibly hierarchies) so that the link count of the final, once-linked source file is reduced to 1 when it is considered below. But in this case (for mv) we don't need to incur the expense of recording the dev/ino => name mapping; all we really need is a lookup, to see if the dev/ino pair has already been copied.
- when using -H and processing a command line argument; that command line argument could be a symlink pointing to another command line argument. With 'cp -H --preserve=link', we hard-link those two destination files.
- likewise for -L except that it applies to all files, not just command line arguments.

Also, with --recursive, record dev/ino of each command-line directory. We'll use that info to detect this problem: cp -R dir dir. \*/

```

if (rename_errno == 0 || x->exchange)
    earlier_file = nullptr;
else if (x->recursive && S_ISDIR (src_mode))
{
    if (command_line_arg)
        earlier_file = remember_copied (dst_relname,
                                         src_sb.st_ino, src_sb.st_dev);
    else
        earlier_file = src_to_dest_lookup (src_sb.st_ino, src_sb.st_dev);
}
else if (x->move_mode && src_sb.st_nlink == 1)
{
    earlier_file = src_to_dest_lookup (src_sb.st_ino, src_sb.st_dev);
}
else if (x->preserve_links
        && !x->hard_link

```

```

    && (1 < src_sb.st_nlink
        || (command_line_arg
            && x->dereference == DEREF_COMMAND_LINE_ARGUMENTS)
        || x->dereference == DEREF_ALWAYS))
    {
        earlier_file = remember_copied (dst_relname,
                                        src_sb.st_ino, src_sb.st_dev);
    }

/* Did we copy this inode somewhere else (in this command line argument)
   and therefore this is a second hard link to the inode? */

if (earlier_file)
{
    /* Avoid damaging the destination file system by refusing to preserve
       hard-linked directories (which are found at least in Netapp snapshot
       directories). */
    if (S_ISDIR (src_mode))
    {
        /* If src_name and earlier_file refer to the same directory entry,
           then warn about copying a directory into itself. */
        if (same_nameat (AT_FDCWD, src_name, dst_dirfd, earlier_file))
        {
            error (0, 0, _("cannot copy a directory, %s, into itself, %s"),
                  quoteaf_n (0, top_level_src_name),
                  quoteaf_n (1, top_level_dst_name));
            *copy_into_self = true;
            goto un_backup;
        }
        else if (same_nameat (dst_dirfd, dst_relname,
                             dst_dirfd, earlier_file))
        {
            error (0, 0, _("warning: source directory %s "
                           "specified more than once"),
                  quoteaf (top_level_src_name));
            /* In move mode, if a previous rename succeeded, then
               we won't be in this path as the source is missing. If the
               rename previously failed, then that has been handled, so
               pretend this attempt succeeded so the source isn't removed. */
            if (x->move_mode && rename_succeeded)
                *rename_succeeded = true;
            /* We only do backups in move mode, and for non directories.
               So just ignore this repeated entry. */
            return true;
        }
        else if (x->dereference == DEREF_ALWAYS
                 || (command_line_arg
                     && x->dereference == DEREF_COMMAND_LINE_ARGUMENTS))
        {
            /* This happens when e.g., encountering a directory for the
               second or subsequent time via symlinks when cp is invoked
               with -R and -L. E.g.,
               rm -rf a b c d; mkdir a b c d; ln -s ..c a; ln -s ..c b;
               cp -RL a b d
            */
        }
    }
    else
    {
        char *earlier = subst_suffix (dst_name, dst_relname,
                                      earlier_file);

```

```

        error (0, 0, _("will not create hard link %s to directory %s"),
               quoteaf_n (0, dst_name), quoteaf_n (1, earlier));
        free (earlier);
        goto un_backup;
    }
}
else
{
    if (! create_hard_link (nullptr, dst_dirfd, earlier_file,
                           dst_name, dst_dirfd, dst_relname,
                           true, x->verbose, dereference))
        goto un_backup;

    return true;
}
}

if (x->move_mode)
{
    if (rename_errno == EEXIST)
        rename_errno = ((renameat (AT_FDCWD, src_name, dst_dirfd, drelname,
                                  x->exchange ? RENAME_EXCHANGE : 0)
                        == 0)
                       ? 0 : errno);

    if (rename_errno == 0)
    {
        if (x->verbose)
            emit_verbose (x->exchange
                          ? _("exchanged %s <-> %s")
                          : _("renamed %s -> %s"),
                          src_name, dst_name, dst_backup);

        if (x->set_security_context)
        {
            /* -Z failures are only warnings currently. */
            (void) set_file_security_ctx (dst_name, true, x);
        }
    }

    if (rename_succeeded)
        *rename_succeeded = true;

    if (command_line_arg && !x->last_file)
    {
        /* Record destination dev/ino/name, so that if we are asked
           to overwrite that file again, we can detect it and fail. */
        /* It's fine to use the _source_ stat buffer (src_sb) to get the
           _destination_ dev/ino, since the rename above can't have
           changed those, and 'mv' always uses lstat.
           We could limit it further by operating
           only on non-directories when !x->exchange. */
        record_file (x->dest_info, dst_relname, &src_sb);
    }
}

return true;
}

/* FIXME: someday, consider what to do when moving a directory into
   itself but when source and destination are on different devices. */

```

```

/* This happens when attempting to rename a directory to a
   subdirectory of itself. */
if (rename_errno == EINVAL)
{
    /* FIXME: this is a little fragile in that it relies on rename(2)
       failing with a specific errno value. Expect problems on
       non-POSIX systems. */
    error (0, 0, _("cannot move %s to a subdirectory of itself, %s"),
           quoteaf_n (0, top_level_src_name),
           quoteaf_n (1, top_level_dst_name));

    /* Note that there is no need to call forget_created here,
       (compare with the other calls in this file) since the
       destination directory didn't exist before. */

    *copy_into_self = true;
    /* FIXME-cleanup: Don't return true here; adjust mv.c accordingly.
       The only caller that uses this code (mv.c) ends up setting its
       exit status to nonzero when copy_into_self is nonzero. */
    return true;
}

/* WARNING: there probably exist systems for which an inter-device
   rename fails with a value of errno not handled here.
   If/as those are reported, add them to the condition below.
   If this happens to you, please do the following and send the output
   to the bug-reporting address (e.g., in the output of cp --help):
   touch k; perl -e 'rename "k","/tmp/k" or print "$!($!+0,)\n"'
   where your current directory is on one partition and /tmp is the other.
   Also, please try to find the E* errno macro name corresponding to
   the diagnostic and parenthesized integer, and include that in your
   e-mail. One way to do that is to run a command like this
   find /usr/include/. -type f \
   | xargs grep 'define.*\<E[A-Z]*\>.*\<18\>' /dev/null
   where you'd replace '18' with the integer in parentheses that
   was output from the perl one-liner above.
   If necessary, of course, change '/tmp' to some other directory. */
if (rename_errno != EXDEV || x->no_copy || x->exchange)
{
    /* There are many ways this can happen due to a race condition.
       When something happens between the initial follow_fstatat and the
       subsequent rename, we can get many different types of errors.
       For example, if the destination is initially a non-directory
       or non-existent, but it is created as a directory, the rename
       fails. If two 'mv' commands try to rename the same file at
       about the same time, one will succeed and the other will fail.
       If the permissions on the directory containing the source or
       destination file are made too restrictive, the rename will
       fail. Etc. */
    char const *quoted_dst_name = quoteaf_n (1, dst_name);
    if (x->exchange)
        error (0, rename_errno, _("cannot exchange %s and %s"),
               quoteaf_n (0, src_name), quoted_dst_name);
    else
        switch (rename_errno)
        {
            case EDQUOT: case EEXIST: case EISDIR: case EMLINK:
            case ENOSPC: case ETXTBSY:
#if ENOTEMPTY != EEXIST
            case ENOTEMPTY:

```

```

#endif
        /* The destination must be the problem. Don't mention
           the source as that is more likely to confuse the user
           than be helpful. */
        error (0, rename_errno, _("cannot overwrite %s"),
               quoted_dst_name);
        break;

    default:
        error (0, rename_errno, _("cannot move %s to %s"),
               quoteaf_n (0, src_name), quoted_dst_name);
        break;
    }

forget_created (src_sb.st_ino, src_sb.st_dev);
return false;
}

/* The rename attempt has failed. Remove any existing destination
   file so that a cross-device 'mv' acts as if it were really using
   the rename syscall. Note both src and dst must both be directories
   or not, and this is enforced above. Therefore we check the src_mode
   and operate on dst_name here as a tighter constraint and also because
   src_mode is readily available here. */
if ((unlinkat (dst_dirfd, drelname,
               S_ISDIR (src_mode) ? AT_REMOVEDIR : 0)
!= 0)
&& errno != ENOENT)
{
    error (0, errno,
           _("inter-device move failed: %s to %s; unable to remove target"),
           quoteaf_n (0, src_name), quoteaf_n (1, dst_name));
forget_created (src_sb.st_ino, src_sb.st_dev);
return false;
}

if (x->verbose && !S_ISDIR (src_mode))
    emit_verbose (_("copied %s -> %s"), src_name, dst_name, dst_backup);
new_dst = true;
}

/* If the ownership might change, or if it is a directory (whose
   special mode bits may change after the directory is created),
   omit some permissions at first, so unauthorized users cannot nip
   in before the file is ready. */
dst_mode_bits = (x->set_mode ? x->mode : src_mode) & CHMOD_MODE_BITS;
omitted_permissions =
(dst_mode_bits
& (x->preserve_ownership ? S_IRWXG | S_IRWXO
: S_ISDIR (src_mode) ? S_IWGRP | S_IWOTH
: 0));
delayed_ok = true;

/* If required, set the default security context for new files.
   Also for existing files this is used as a reference
   when copying the context with --preserve=context.
   FIXME: Do we need to consider dst_mode_bits here? */
if (!set_process_security_ctx (src_name, dst_name, src_mode, new_dst, x))
    return false;

```

```

if (S_ISDIR (src_mode))
{
    struct dir_list *dir;

    /* If this directory has been copied before during the
       recursion, there is a symbolic link to an ancestor
       directory of the symbolic link. It is impossible to
       continue to copy this, unless we've got an infinite file system. */

    if (is_ancestor (&src_sb, ancestors))
    {
        error (0, 0, _("cannot copy cyclic symbolic link %s"),
               quoteaf (src_name));
        goto un_backup;
    }

    /* Insert the current directory in the list of parents. */

    dir = alloca (sizeof *dir);
    dir->parent = ancestors;
    dir->ino = src_sb.st_ino;
    dir->dev = src_sb.st_dev;

    if (new_dst || !S_ISDIR (dst_sb.st_mode))
    {
        /* POSIX says mkdir's behavior is implementation-defined when
           (src_mode & ~S_IRWXUGO) != 0. However, common practice is
           to ask mkdir to copy all the CHMOD_MODE_BITS, letting mkdir
           decide what to do with S_ISUID | S_ISGID | S_ISVTX. */
        mode_t mode = dst_mode_bits & ~omitted_permissions;
        if (mkdirat (dst_dirfd, drelname, mode) != 0)
        {
            error (0, errno, _("cannot create directory %s"),
                   quoteaf (dst_name));
            goto un_backup;
        }

        /* We need search and write permissions to the new directory
           for writing the directory's contents. Check if these
           permissions are there. */

        if (fstatat (dst_dirfd, drelname, &dst_sb, AT_SYMLINK_NOFOLLOW) != 0)
        {
            error (0, errno, _("cannot stat %s"), quoteaf (dst_name));
            goto un_backup;
        }
        else if ((dst_sb.st_mode & S_IRWXU) != S_IRWXU)
        {
            /* Make the new directory searchable and writable. */

            dst_mode = dst_sb.st_mode;
            restore_dst_mode = true;

            if (!chmodat (dst_dirfd, drelname, dst_mode | S_IRWXU) != 0)
            {
                error (0, errno,_("setting permissions for %s"),
                      quoteaf (dst_name));
                goto un_backup;
            }
        }
    }
}

```

```

/* Record the created directory's inode and device numbers into
   the search structure, so that we can avoid copying it again.
   Do this only for the first directory that is created for each
   source command line argument. */
if (!first_dir_created_per_command_line_arg)
{
    remember_copied (dst_relname, dst_sb.st_ino, dst_sb.st_dev);
    *first_dir_created_per_command_line_arg = true;
}

if (x->verbose)
{
    if (x->move_mode)
        printf (_("created directory %s\n"), quoteaf (dst_name));
    else
        emit_verbose ("%s -> %s", src_name, dst_name, nullptr);
}
else
{
    omitted_permissions = 0;

/* For directories, the process global context could be reset for
   descendants, so use it to set the context for existing dirs here.
   This will also give earlier indication of failure to set ctx. */
if (x->set_security_context || x->preserve_security_context)
    if (!set_file_security_ctx (dst_name, false, x))
    {
        if (x->require_preserve_context)
            goto un_backup;
    }
}

/* Decide whether to copy the contents of the directory. */
if (x->one_file_system && parent && parent->st_dev != src_sb.st_dev)
{
    /* Here, we are crossing a file system boundary and cp's -x option
       is in effect: so don't copy the contents of this directory. */
}
else
{
    /* Copy the contents of the directory. Don't just return if
       this fails -- otherwise, the failure to read a single file
       in a source directory would cause the containing destination
       directory not to have owner perms set properly. */
    delayed_ok = copy_dir (src_name, dst_name, dst_dirfd, dst_relname,
                           new_dst, &src_sb, dir, x,
                           first_dir_created_per_command_line_arg,
                           copy_into_self);
}
else if (x->symbolic_link)
{
    dest_is_symlink = true;
    if (*src_name != '/')
    {
        /* Check that DST_NAME denotes a file in the current directory. */
        struct stat dot_sb;
        struct stat dst_parent_sb;

```

```

char *dst_parent;
bool in_current_dir;

dst_parent = dir_name (dst_relname);

in_current_dir = ((dst_dirfd == AT_FDCWD && STREQ (".", dst_parent))
    /* If either stat call fails, it's ok not to report
     the failure and say dst_name is in the current
     directory. Other things will fail later. */
    || stat (".", &dot_sb) != 0
    || (fstatat (dst_dirfd, dst_parent, &dst_parent_sb,
        0) != 0)
    || psame_inode (&dot_sb, &dst_parent_sb));
free (dst_parent);

if (!in_current_dir)
{
    error (0, 0,
        _("%s: can make relative symbolic links only in current directory"),
        quoteef (dst_name));
    goto un_backup;
}
}

int err = force_symlinkat (src_name, dst_dirfd, dst_relname,
    x->unlink_dest_after_failed_open, -1);
if (0 < err)
{
    error (0, err, _("cannot create symbolic link %s to %s"),
        quoteaf_n (0, dst_name), quoteaf_n (1, src_name));
    goto un_backup;
}
}

/* POSIX 2008 states that it is implementation-defined whether
link() on a symlink creates a hard-link to the symlink, or only
to the referent (effectively dereferencing the symlink) (POSIX
2001 required the latter behavior, although many systems provided
the former). Yet cp, invoked with '--link --no-dereference',
should not follow the link. We can approximate the desired
behavior by skipping this hard-link creating block and instead
copying the symlink, via the 'S_ISLNK'- copying code below.

Note gnulib's linkat module, guarantees that the symlink is not
dereferenced. However its emulation currently doesn't maintain
timestamps or ownership so we only call it when we know the
emulation will not be needed. */
else if (x->hard_link
    && !( CAN_HARDLINK_SYMLINKS && S_ISLNK (src_mode)
        && x->dereference == DEREF_NEVER))
{
    bool replace = (x->unlink_dest_after_failed_open
        || x->interactive == I_ASK_USER);
    if (!create_hard_link (src_name, AT_FDCWD, src_name,
        dst_name, dst_dirfd, dst_relname,
        replace, false, dereference))
        goto un_backup;
}
else if (S_ISREG (src_mode)
    || (x->copy_as_regular && !S_ISLNK (src_mode)))

```

```

{
copied_as_regular = true;
/* POSIX says the permission bits of the source file must be
   used as the 3rd argument in the open call. Historical
   practice passed all the source mode bits to 'open', but the extra
   bits were ignored, so it should be the same either way.

This call uses DST_MODE_BITS, not SRC_MODE. These are
normally the same, and the exception (where x->set_mode) is
used only by 'install', which POSIX does not specify and
where DST_MODE_BITS is what's wanted. */
if (! copy_reg (src_name, dst_name, dst_dirfd, dst_relname,
               x, dst_mode_bits & S_IRWXUGO,
               omitted_permissions, &new_dst, &src_sb))
    goto un_backup;
}
else if (S_ISFIFO (src_mode))
{
/* Use mknodat, rather than mkfifoat, because the former preserves
   the special mode bits of a fifo on Solaris 10, while mkfifoat
   does not. But fall back on mkfifoat, because on some BSD systems,
   mknodat always fails when asked to create a FIFO. */
mode_t mode = src_mode & ~omitted_permissions;
if (mknodat (dst_dirfd, dst_relname, mode, 0) != 0)
    if (mkfifoat (dst_dirfd, dst_relname, mode & ~S_IFIFO) != 0)
    {
        error (0, errno, _("cannot create fifo %s"), quoteaf (dst_name));
        goto un_backup;
    }
}
else if (S_ISBLK (src_mode) || S_ISCHR (src_mode) || S_ISSOCK (src_mode))
{
mode_t mode = src_mode & ~omitted_permissions;
if (mknodat (dst_dirfd, dst_relname, mode, src_sb.st_rdev) != 0)
{
    error (0, errno, _("cannot create special file %s"),
           quoteaf (dst_name));
    goto un_backup;
}
}
else if (S_ISLNK (src_mode))
{
char *src_link_val = areadlink_with_size (src_name, src_sb.st_size);
dest_is_symlink = true;
if (src_link_val == nullptr)
{
    error (0, errno, _("cannot read symbolic link %s"),
           quoteaf (src_name));
    goto un_backup;
}

int symlink_err = force_symlinkat (src_link_val, dst_dirfd, dst_relname,
                                   x->unlink_dest_after_failed_open, -1);
if (0 < symlink_err && x->update && !new_dst && S_ISLNK (dst_sb.st_mode)
    && dst_sb.st_size == strlen (src_link_val))
{
/* See if the destination is already the desired symlink.
   FIXME: This behavior isn't documented, and seems wrong
   in some cases, e.g., if the destination symlink has the
   wrong ownership, permissions, or timestamps. */

```

```

char *dest_link_val =
    areadlinkat_with_size (dst_dirfd, dst_relname, dst_sb.st_size);
if (dest_link_val)
{
    if (STREQ (dest_link_val, src_link_val))
        symlink_err = 0;
    free (dest_link_val);
}
free (src_link_val);
if (0 < symlink_err)
{
    error (0, symlink_err, _("cannot create symbolic link %s"),
           quoteaf (dst_name));
    goto un_backup;
}

if (x->preserve_security_context)
    restore_default_fscreatecon_or_die ();

if (x->preserve_ownership)
{
    /* Preserve the owner and group of the just-'copied'
       symbolic link, if possible. */
    if (HAVE_LCHOWN
        && (lchownat (dst_dirfd, dst_relname,
                      src_sb.st_uid, src_sb.st_gid)
             != 0)
        && ! chown_failure_ok (x))
    {
        error (0, errno, _("failed to preserve ownership for %s"),
               dst_name);
        if (x->require_preserve)
            goto un_backup;
    }
    else
    {
        /* Can't preserve ownership of symlinks.
           FIXME: maybe give a warning or even error for symlinks
           in directories with the sticky bit set -- there, not
           preserving owner/group is a potential security problem. */
    }
}
else
{
    error (0, 0, _("'%s has unknown file type"), quoteaf (src_name));
    goto un_backup;
}

/* With -Z or --preserve=context, set the context for existing files.
   Note this is done already for copy_reg() for reasons described therein. */
if (!new_dst && !x->copy_as_regular && !S_ISDIR (src_mode)
    && (x->set_security_context || x->preserve_security_context))
{
    if (! set_file_security_ctx (dst_name, false, x))
    {
        if (x->require_preserve_context)
            goto un_backup;
    }
}

```

```

    }

if (command_line_arg && x->dest_info)
{
/* Now that the destination file is very likely to exist,
   add its info to the set. */
struct stat sb;
if (fstatat (dst_dirfd, drelname, &sb, AT_SYMLINK_NOFOLLOW) == 0)
    record_file (x->dest_info, dst_relnname, &sb);
}

/* If we've just created a hard-link due to cp's --link option,
   we're done. */
if (x->hard_link && ! S_ISDIR (src_mode)
    && !( CAN_HARDLINK_SYMLINKS && S_ISLNK (src_mode)
          && x->dereference == DEREF_NEVER))
    return delayed_ok;

if (copied_as_regular)
    return delayed_ok;

/* POSIX says that 'cp -p' must restore the following:
   - permission bits
   - setuid, setgid bits
   - owner and group
   If it fails to restore any of those, we may give a warning but
   the destination must not be removed.
   FIXME: implement the above. */

/* Adjust the times (and if possible, ownership) for the copy.
   chown turns off set[ug]id bits for non-root,
   so do the chmod last. */

if (x->preserve_timestamps)
{
    struct timespec timespec[2];
    timespec[0] = get_stat_atime (&src_sb);
    timespec[1] = get_stat_mtime (&src_sb);

    int utimensat_flags = dest_is_symlink ? AT_SYMLINK_NOFOLLOW : 0;
    if (utimensat (dst_dirfd, drelname, timespec, utimensat_flags) != 0)
    {
        error (0, errno, _("preserving times for %s"), quoteaf (dst_name));
        if (x->require_preserve)
            return false;
    }
}

/* Avoid calling chown if we know it's not necessary. */
if (!dest_is_symlink && x->preserve_ownership
    && (new_dst || !SAME_OWNER_AND_GROUP (src_sb, dst_sb)))
{
    switch (set_owner (x, dst_name, dst_dirfd, drelname, -1,
                      &src_sb, new_dst, &dst_sb))
    {
        case -1:
            return false;

        case 0:
            src_mode &= ~ (S_ISUID | S_ISGID | S_ISVTX);
    }
}

```

```

        break;
    }
}

/* Set xattrs after ownership as changing owners will clear capabilities. */
if (x->preserve_xattr && ! copy_attr (src_name, -1, dst_name, -1, x)
    && x->require_preserve_xattr)
    return false;

/* The operations beyond this point may dereference a symlink. */
if (dest_is_symlink)
    return delayed_ok;

set_author (dst_name, -1, &src_sb);

if (x->preserve_mode || x->move_mode)
{
    if (copy_acl (src_name, -1, dst_name, -1, src_mode) != 0
        && x->require_preserve)
        return false;
}
else if (x->set_mode)
{
    if (set_acl (dst_name, -1, x->mode) != 0)
        return false;
}
else if (x->explicit_no_preserve_mode && new_dst)
{
    int default_permissions = S_ISDIR (src_mode) || S_ISSOCK (src_mode)
        ? S_IRWXUGO : MODE_RW_UGO;
    dst_mode = dst_sb.st_mode;
    if (S_ISDIR (src_mode)) /* Keep set-group-ID for directories. */
        default_permissions |= (dst_mode & S_ISGID);
    if (set_acl (dst_name, -1, default_permissions & ~cached_umask ()) != 0)
        return false;
}
else
{
    if (omitted_permissions)
    {
        omitted_permissions &= ~ cached_umask ();

        if (omitted_permissions && !restore_dst_mode)
        {
            /* Permissions were deliberately omitted when the file
               was created due to security concerns. See whether
               they need to be re-added now. It'd be faster to omit
               the lstat, but deducing the current destination mode
               is tricky in the presence of implementation-defined
               rules for special mode bits. */
            if (new_dst && (fstatat (dst_dirfd, drelname, &dst_sb,
                                         AT_SYMLINK_NOFOLLOW)
                           != 0))
            {
                error (0, errno, _("cannot stat %s"), quoteaf (dst_name));
                return false;
            }
            dst_mode = dst_sb.st_mode;
            if (omitted_permissions & ~dst_mode)
                restore_dst_mode = true;
        }
    }
}

```

```

        }

    }

if (restore_dst_mode)
{
    if (!chmodat (dst_dirfd, drelname, dst_mode | omitted_permissions)
        != 0)
    {
        error (0, errno, _("preserving permissions for %s"),
               quoteaf (dst_name));
        if (x->require_preserve)
            return false;
    }
}

return delayed_ok;

un_backup:

if (x->preserve_security_context)
    restore_default_fscREATECON_OR_DIE ();

/* We have failed to create the destination file.
   If we've just added a dev/ino entry via the remember_copied
   call above (i.e., unless we've just failed to create a hard link),
   remove the entry associating the source dev/ino with the
   destination file name, so we don't try to 'preserve' a link
   to a file we didn't create. */
if (earlier_file == nullptr)
    forget_created (src_sb.st_ino, src_sb.st_dev);

if (dst_backup)
{
    char const *dst_rebackup = &dst_backup[dst_relname - dst_name];
    if (renameat (dst_dirfd, dst_rebackup, dst_dirfd, drename) != 0)
        error (0, errno, _("cannot un-backup %s"), quoteaf (dst_name));
    else
    {
        if (x->verbose)
            printf (_("%s -> %s (unbackup)\n"),
                    quoteaf_n (0, dst_backup), quoteaf_n (1, dst_name));
    }
}
return false;
}

static void
valid_options (const struct cp_options *co)
{
affirm (VALID_BACKUP_TYPE (co->backup_type));
affirm (VALID_SPARSE_MODE (co->sparse_mode));
affirm (VALID_REFLINK_MODE (co->reflink_mode));
affirm (!(co->hard_link && co->symbolic_link));
affirm (!(
            (co->reflink_mode == REFLINK_ALWAYS
             && co->sparse_mode != SPARSE_AUTO)));
}

/* Copy the file SRC_NAME to the file DST_NAME aka DST_DIRFD+DST_RELNAME.
```

If NONEXISTENT\_DST is positive, DST\_NAME does not exist even as a dangling symlink; if negative, it does not exist except possibly as a dangling symlink; if zero, its existence status is unknown.

OPTIONS summarizes the command-line options.

Set \*COPY\_INTO\_SELF if SRC\_NAME is a parent of (or the same as) DST\_NAME; otherwise, set clear it.

If X->move\_mode, set \*RENAME\_SUCCEEDED according to whether the source was simply renamed to the destination.

Return true if successful. \*/

```

extern bool
copy (char const *src_name, char const *dst_name,
      int dst_dirfd, char const *dst_relname,
      int nonexistent_dst, const struct cp_options *options,
      bool *copy_into_self, bool *rename_succeeded)
{
    valid_options (options);

    /* Record the file names: they're used in case of error, when copying
       a directory into itself. I don't like to make these tools do "any"
       extra work in the common case when that work is solely to handle
       exceptional cases, but in this case, I don't see a way to derive the
       top level source and destination directory names where they're used.
       An alternative is to use COPY_INTO_SELF and print the diagnostic
       from every caller -- but I don't want to do that. */
    top_level_src_name = src_name;
    top_level_dst_name = dst_name;

    bool first_dir_created_per_command_line_arg = false;
    return copy_internal (src_name, dst_name, dst_dirfd, dst_relname,
                          nonexistent_dst, nullptr, nullptr,
                          options, true,
                          &first_dir_created_per_command_line_arg,
                          copy_into_self, rename_succeeded);
}

/* Set *X to the default options for a value of type struct cp_options. */

extern void
cp_options_default (struct cp_options *x)
{
    memset (x, 0, sizeof *x);
    #ifdef PRIV_FILE_CHOWN
    {
        priv_set_t *pset = priv_allocset ();
        if (!pset)
            xalloc_die ();
        if (getppriv (PRIV_EFFECTIVE, pset) == 0)
        {
            x->chown_privileges = priv_ismember (pset, PRIV_FILE_CHOWN);
            x->owner_privileges = priv_ismember (pset, PRIV_FILE_OWNER);
        }
        priv_freeset (pset);
    }
    #else
    x->chown_privileges = x->owner_privileges = (geteuid () == ROOT_UID);
    #endif
    x->rename_errno = -1;
}

```

```

/* Return true if it's OK for chown to fail, where errno is
the error number that chown failed with and X is the copying
option set. */

extern bool
chown_failure_ok (struct cp_options const *x)
{
/* If non-root uses -p, it's ok if we can't preserve ownership.
But root probably wants to know, e.g. if NFS disallows it,
or if the target system doesn't support file ownership.

Treat EACCES like EPERM and EINVAL to work around a bug in Linux
CIFS <https://bugs.gnu.org/65599>. Although this means coreutils
will ignore EACCES errors that it should report, problems should
occur only when some other process is racing with coreutils and
coreutils is not immune to races anyway. */

return ((errno == EPERM || errno == EINVAL || errno == EACCES)
       && !x->chown_privileges);
}

/* Similarly, return true if it's OK for chmod and similar operations
to fail, where errno is the error number that chmod failed with and
X is the copying option set. */

static bool
owner_failure_ok (struct cp_options const *x)
{
return ((errno == EPERM || errno == EINVAL || errno == EACCES)
       && !x->owner_privileges);
}

/* Return the user's umask, caching the result.

FIXME: If the destination's parent directory has has a default ACL,
some operating systems (e.g., GNU/Linux's "POSIX" ACLs) use that
ACL's mask rather than the process umask. Currently, the callers
of cached_umask incorrectly assume that this situation cannot occur. */
extern mode_t
cached_umask (void)
{
static mode_t mask = (mode_t) -1;
if (mask == (mode_t) -1)
{
    mask = umask (0);
    umask (mask);
}
return mask;
}
"""

"error_category": "z/OS-Specific Compilation Error",
"error": "EDC5130I: Incompatible or undefined macro/method usage on z/OS.",
"correct_code":
"""

/* copy.c -- core functions for copying files and directories
Copyright (C) 1989-2024 Free Software Foundation, Inc.
```

This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or

(at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License  
along with this program. If not, see <<https://www.gnu.org/licenses/>>. \*/

/\* Extracted from cp.c and librarified by Jim Meyering. \*/

```
#include <config.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <selinux/selinux.h>

#if HAVE_HURD_H
# include <hurd.h>
#endif
#if HAVE_PRIV_H
# include <priv.h>
#endif

#include "system.h"
#include "acl.h"
#include "alignalloc.h"
#include "assure.h"
#include "backupfile.h"
#include "buffer-lcm.h"
#include "canonicalize.h"
#include "copy.h"
#include "cp-hash.h"
#include "fadvise.h"
#include "fcntl--.h"
#include "file-set.h"
#include "filemode.h"
#include "filenamecat.h"
#include "force-link.h"
#include "full-write.h"
#include "hash.h"
#include "hash-triple.h"
#include "ignore-value.h"
#include "ioblksize.h"
#include "quote.h"
#include "renameatu.h"
#include "root-uid.h"
#include "same.h"
#include "savedir.h"
#include "stat-size.h"
#include "stat-time.h"
#include "utimecmp.h"
#include "utimens.h"
#include "write-any-file.h"
#include "areadlink.h"
#include "yesno.h"
#include "selinux.h"

#ifndef USE_XATTR
```

```

#define USE_XATTR false
#endif

#if USE_XATTR
#include <attr/error_context.h>
#include <attr/libattr.h>
#include <stdarg.h>
#include "verror.h"
#endif

#if HAVE_LINUX_FALLOC_H
#include <linux/falloc.h>
#endif

/* See HAVE_FALLOCATE workaround when including this file. */
#ifndef HAVE_LINUX_FS_H
#include <linux/fs.h>
#endif

#ifndef FICLONE && defined __linux__
#define FICLONE _IOW(0x94, 9, int)
#endif

#ifndef HAVE_FCLONEFILEAT && !USE_XATTR
#include <sys/clonefile.h>
#endif

#ifndef USE_ACL
#define USE_ACL 0
#endif

#define SAME_OWNER(A, B) ((A).st_uid == (B).st_uid)
#define SAME_GROUP(A, B) ((A).st_gid == (B).st_gid)
#define SAME_OWNER_AND_GROUP(A, B) (SAME_OWNER(A, B) && SAME_GROUP(A,
B))

/* LINK_FOLLOWING_SYMLINKS is tri-state; if it is -1, we don't know
how link() behaves, so assume we can't hardlink symlinks in that case. */
#ifndef (defined HAVE_LINKAT && !LINKAT_SYMLINK_NOTSUP) || !
LINK_FOLLOWING_SYMLINKS
#define CAN_HARDLINK_SYMLINKS 1
#else
#define CAN_HARDLINK_SYMLINKS 0
#endif

#ifndef __MVS__
#include "zos-io.h"
#endif

struct dir_list
{
    struct dir_list *parent;
    ino_t ino;
    dev_t dev;
};

/* Initial size of the cp.dest_info hash table. */
#define DEST_INFO_INITIAL_CAPACITY 61

static bool copy_internal (char const *src_name, char const *dst_name,

```

```

        int dst_dirfd, char const *dst_relname,
        int nonexistent_dst, struct stat const *parent,
        struct dir_list *ancestors,
        const struct cp_options *x,
        bool command_line_arg,
        bool *first_dir_created_per_command_line_arg,
        bool *copy_into_self,
        bool *rename_succeeded);
static bool owner_failure_ok (struct cp_options const *x);

/* Pointers to the file names: they're used in the diagnostic that is issued
when we detect the user is trying to copy a directory into itself. */
static char const *top_level_src_name;
static char const *top_level_dst_name;

enum copy_debug_val
{
COPY_DEBUG_UNKNOWN,
COPY_DEBUG_NO,
COPY_DEBUG_YES,
COPY_DEBUG_EXTERNAL,
COPY_DEBUG_EXTERNAL_INTERNAL,
COPY_DEBUG_AVOIDED,
COPY_DEBUG_UNSUPPORTED,
};

/* debug info about the last file copy. */
static struct copy_debug
{
enum copy_debug_val offload;
enum copy_debug_val reflink;
enum copy_debug_val sparse_detection;
} copy_debug;

static const char*
copy_debug_string (enum copy_debug_val debug_val)
{
switch (debug_val)
{
case COPY_DEBUG_NO: return "no";
case COPY_DEBUG_YES: return "yes";
case COPY_DEBUG_AVOIDED: return "avoided";
case COPY_DEBUG_UNSUPPORTED: return "unsupported";
default: return "unknown";
}
}

static const char*
copy_debug_sparse_string (enum copy_debug_val debug_val)
{
switch (debug_val)
{
case COPY_DEBUG_NO: return "no";
case COPY_DEBUG_YES: return "zeros";
case COPY_DEBUG_EXTERNAL: return "SEEK_HOLE";
case COPY_DEBUG_EXTERNAL_INTERNAL: return "SEEK_HOLE + zeros";
default: return "unknown";
}
}

```

```

/* Print --debug output on standard output. */
static void
emit_debug (const struct cp_options *x)
{
if (!x->hard_link && !x->symbolic_link && x->data_copy_required)
    printf ("copy offload: %s, reflink: %s, sparse detection: %s\n",
            copy_debug_string (copy_debug.offload),
            copy_debug_string (copy_debug.reflink),
            copy_debug_sparse_string (copy_debug.sparse_detection));
}

#ifndef DEV_FD_MIGHT_BE_CHR
#define DEV_FD_MIGHT_BE_CHR false
#endif

/* Act like fstat (DIRFD, FILENAME, ST, FLAGS), except when following
symbolic links on Solaris-like systems, treat any character-special
device like /dev/fd/0 as if it were the file it is open on. */
static int
follow_fstatat (int dirfd, char const *filename, struct stat *st, int flags)
{
int result = fstatat (dirfd, filename, st, flags);

if (DEV_FD_MIGHT_BE_CHR && result == 0 && !(flags & AT_SYMLINK_NOFOLLOW)
    && S_ISCHR (st->st_mode))
{
    static dev_t stdin_rdev;
    static signed char stdin_rdev_status;
    if (stdin_rdev_status == 0)
    {
        struct stat stdin_st;
        if (stat ("/dev/stdin", &stdin_st) == 0 && S_ISCHR (stdin_st.st_mode)
            && minor (stdin_st.st_rdev) == STDIN_FILENO)
        {
            stdin_rdev = stdin_st.st_rdev;
            stdin_rdev_status = 1;
        }
        else
            stdin_rdev_status = -1;
    }
    if (0 < stdin_rdev_status && major (stdin_rdev) == major (st->st_rdev))
        result = fstat (minor (st->st_rdev), st);
}
}

return result;
}

/* Attempt to punch a hole to avoid any permanent
speculative preallocation on file systems such as XFS.
Return values as per fallocate(2) except ENOSYS etc. are ignored. */

static int
punch_hole (int fd, off_t offset, off_t length)
{
int ret = 0;
/* +0 is to work around older <linux/fs.h> defining HAVE_FALLOCATE to empty. */
#if HAVE_FALLOCATE + 0
# if defined FALLOC_FL_PUNCH_HOLE && defined FALLOC_FL_KEEP_SIZE
ret = fallocate (fd, FALLOC_FL_PUNCH_HOLE | FALLOC_FL_KEEP_SIZE,
                offset, length);

```

```

if (ret < 0 && (is_ENOTSUP (errno) || errno == ENOSYS))
    ret = 0;
#endif
#endif
return ret;
}

/* Create a hole at the end of a file,
   avoiding preallocation if requested. */

static bool
create_hole (int fd, char const *name, bool punch_holes, off_t size)
{
off_t file_end = lseek (fd, size, SEEK_CUR);

if (file_end < 0)
{
    error (0, errno, _("cannot lseek %s"), quoteaf (name));
    return false;
}

/* Some file systems (like XFS) preallocate when write extending a file.
   I.e., a previous write() may have preallocated extra space
   that the seek above will not discard. A subsequent write() could
   then make this allocation permanent. */
if (punch_holes && punch_hole (fd, file_end - size, size) < 0)
{
    error (0, errno, _("error deallocating %s"), quoteaf (name));
    return false;
}

return true;
}

/* Whether an errno value ERR, set by FICLONE or copy_file_range,
   indicates that the copying operation has terminally failed, even
   though it was invoked correctly (so that, e.g, EBADF cannot occur)
   and even though !is_CLONENOTSUP (ERR). */

static bool
is_terminal_error (int err)
{
return err == EIO || err == ENOMEM || err == ENOSPC || err == EDQUOT;
}

/* Similarly, whether ERR indicates that the copying operation is not
   supported or allowed for this file or process, even though the
   operation was invoked correctly. */

static bool
is_CLONENOTSUP (int err)
{
return err == ENOSYS || err == ENOTTY || is_ENOTSUP (err)
    || err == EINVAL || err == EBADF
    || err == EXDEV || err == ETXTBSY
    || err == EPERM || err == EACCES;
}

```

```

/* Copy the regular file open on SRC_FD/SRC_NAME to DST_FD/DST_NAME,
honoring the MAKE_HOLES setting and using the BUF_SIZE-byte buffer
*ABUF for temporary storage, allocating it lazily if *ABUF is null.
Copy no more than MAX_N_READ bytes.
Return true upon successful completion;
print a diagnostic and return false upon error.
Note that for best results, BUF should be "well"-aligned.
Set *LAST_WRITE_MADE_HOLE to true if the final operation on
DEST_FD introduced a hole. Set *TOTAL_N_READ to the number of
bytes read. */
static bool
sparse_copy (int src_fd, int dest_fd, char **abuf, size_t buf_size,
             size_t hole_size, bool punch_holes, bool allow_relink,
             char const *src_name, char const *dst_name,
             uintmax_t max_n_read, off_t *total_n_read,
             bool *last_write_made_hole)
{
    *last_write_made_hole = false;
    *total_n_read = 0;

    if (copy_debug.sparse_detection == COPY_DEBUG_UNKNOWN)
        copy_debug.sparse_detection = hole_size ? COPY_DEBUG_YES : COPY_DEBUG_NO;
    else if (hole_size && copy_debug.sparse_detection == COPY_DEBUG_EXTERNAL)
        copy_debug.sparse_detection = COPY_DEBUG_EXTERNAL_INTERNAL;

    /* If not looking for holes, use copy_file_range if functional,
       but don't use if reflink disallowed as that may be implicit. */
    if (!hole_size && allow_relink)
        while (max_n_read)
    {
        /* Copy at most COPY_MAX bytes at a time; this is min
           (SSIZE_MAX, SIZE_MAX) truncated to a value that is
           surely aligned well. */
        ssize_t copy_max = MIN (SSIZE_MAX, SIZE_MAX) >> 30 << 30;
        ssize_t n_copied = copy_file_range (src_fd, nullptr, dest_fd, nullptr,
                                           MIN (max_n_read, copy_max), 0);
        if (n_copied == 0)
        {
            /* copy_file_range incorrectly returns 0 when reading from
               the proc file system on the Linux kernel through at
               least 5.6.19 (2020), so fall back on 'read' if the
               input file seems empty. */
            if (*total_n_read == 0)
                break;
            copy_debug.offload = COPY_DEBUG_YES;
            return true;
        }
        if (n_copied < 0)
        {
            copy_debug.offload = COPY_DEBUG_UNSUPPORTED;

            /* Consider operation unsupported only if no data copied.
               For example, EPERM could occur if copy_file_range not enabled
               in seccomp filters, so retry with a standard copy. EPERM can
               also occur for immutable files, but that would only be in the
               edge case where the file is made immutable after creating,
               in which case the (more accurate) error is still shown. */
            if (*total_n_read == 0 && is_CLONENOTSUP (errno))
                break;
        }
    }
}

```

```

/* ENOENT was seen sometimes across CIFS shares, resulting in
no data being copied, but subsequent standard copies succeed. */
if (*total_n_read == 0 && errno == ENOENT)
break;

if (errno == EINTR)
n_copied = 0;
else
{
    error (0, errno, _("error copying %s to %s"),
        quoteaf_n (0, src_name), quoteaf_n (1, dst_name));
    return false;
}
copy_debug.offload = COPY_DEBUG_YES;
max_n_read -= n_copied;
*total_n_read += n_copied;
}
else
copy_debug.offload = COPY_DEBUG_AVOIDED;

bool make_hole = false;
off_t psize = 0;

while (max_n_read)
{
if (!*abuf)
*abuf = xalignalloc (getpagesize (), buf_size);
char *buf = *abuf;
ssize_t n_read = read (src_fd, buf, MIN (max_n_read, buf_size));
if (n_read < 0)
{
if (errno == EINTR)
continue;
error (0, errno, _("error reading %s"), quoteaf (src_name));
return false;
}
if (n_read == 0)
break;
max_n_read -= n_read;
*total_n_read += n_read;

/* Loop over the input buffer in chunks of hole_size. */
size_t csize = hole_size ? hole_size : buf_size;
char *cbuf = buf;
char *pbuf = buf;

while (n_read)
{
bool prev_hole = make_hole;
csize = MIN (csize, n_read);

if (hole_size && csize)
make_hole = is_nul (cbuf, csize);

bool transition = (make_hole != prev_hole) && psize;
bool last_chunk = (n_read == csize && ! make_hole) || ! csize;

if (transition || last_chunk)

```

```

{
if (! transition)
    psize += csiz;

if (! prev_hole)
{
    if (full_write (dest_fd, pbuf, psize) != psize)
    {
        error (0, errno, _("error writing %s"),
               quoteaf (dst_name));
        return false;
    }
}
else
{
    if (! create_hole (dest_fd, dst_name, punch_holes, psize))
        return false;
}

pbuf = cbuf;
psize = csiz;

if (last_chunk)
{
    if (! csiz)
        n_read = 0; /* Finished processing buffer. */

    if (transition)
        csiz = 0; /* Loop again to deal with last chunk. */
    else
        psize = 0; /* Reset for next read loop. */
}
else /* Coalesce writes/seeks. */
{
    if (ckd_add (&psize, psiz, csiz))
    {
        error (0, 0, _("overflow reading %s"), quoteaf (src_name));
        return false;
    }
}

n_read -= csiz;
cbuf += csiz;
}

/*last_write_made_hole = make_hole;

/* It's tempting to break early here upon a short read from
   a regular file. That would save the final read syscall
   for each file. Unfortunately that doesn't work for
   certain files in /proc or /sys with linux kernels. */
}

/* Ensure a trailing hole is created, so that subsequent
   calls of sparse_copy() start at the correct offset. */
if (make_hole && ! create_hole (dest_fd, dst_name, punch_holes, psize))
    return false;
else
    return true;

```

```

}

/* Perform the O(1) btrfs clone operation, if possible.
Upon success, return 0. Otherwise, return -1 and set errno. */
static inline int
clone_file (int dest_fd, int src_fd)
{
#ifndef FICLONE
return ioctl (dest_fd, FICLONE, src_fd);
#else
(void) dest_fd;
(void) src_fd;
errno = ENOTSUP;
return -1;
#endif
}

/* Write N_BYTES zero bytes to file descriptor FD. Return true if successful.
Upon write failure, set errno and return false. */
static bool
write_zeros (int fd, off_t n_bytes)
{
static char *zeros;
static size_t nz = IO_BUFSIZE;

/* Attempt to use a relatively large calloc'd source buffer for
   efficiency, but if that allocation fails, resort to a smaller
   statically allocated one. */
if (zeros == nullptr)
{
    {
static char fallback[1024];
zeros = calloc (nz, 1);
if (zeros == nullptr)
    {
zeros = fallback;
nz = sizeof fallback;
    }
}
}

while (n_bytes)
{
size_t n = MIN (nz, n_bytes);
if ((full_write (fd, zeros, n)) != n)
    return false;
n_bytes -= n;
}

return true;
}

#endif /* _BTRFS_H */

#endif /* _LIBBTRFS_H */

```

```

Return true if successful, false (with a diagnostic) otherwise. */

static bool
lseek_copy (int src_fd, int dest_fd, char **abuf, size_t buf_size,
            size_t hole_size, off_t ext_start, off_t src_total_size,
            enum Sparse_type sparse_mode,
            bool allow_relink,
            char const *src_name, char const *dst_name)
{
off_t last_ext_start = 0;
off_t last_ext_len = 0;
off_t dest_pos = 0;
bool wrote_hole_at_eof = true;

copy_debug.sparse_detection = COPY_DEBUG_EXTERNAL;

while (0 <= ext_start)
{
off_t ext_end = lseek (src_fd, ext_start, SEEK_HOLE);
if (ext_end < 0)
{
if (errno != ENXIO)
    goto cannot_lseek;
ext_end = src_total_size;
if (ext_end <= ext_start)
{
/* The input file grew; get its current size. */
src_total_size = lseek (src_fd, 0, SEEK_END);
if (src_total_size < 0)
    goto cannot_lseek;

/* If the input file shrank after growing, stop copying. */
if (src_total_size <= ext_start)
    break;

ext_end = src_total_size;
}
}
/* If the input file must have grown, increase its measured size. */
if (src_total_size < ext_end)
    src_total_size = ext_end;

if (lseek (src_fd, ext_start, SEEK_SET) < 0)
    goto cannot_lseek;

wrote_hole_at_eof = false;
off_t ext_hole_size = ext_start - last_ext_start - last_ext_len;

if (ext_hole_size)
{
if (sparse_mode != SPARSE_NEVER)
{
if (! create_hole (dest_fd, dst_name,
                  sparse_mode == SPARSE_ALWAYS,
                  ext_hole_size))
    return false;
wrote_hole_at_eof = true;
}
else
{

```

```

/* When not inducing holes and when there is a hole between
   the end of the previous extent and the beginning of the
   current one, write zeros to the destination file. */
if (! write_zeros (dest_fd, ext_hole_size))
{
{
    error (0, errno, _("'%s: write failed"),
          quoteaf (dst_name));
    return false;
}
}

off_t ext_len = ext_end - ext_start;
last_ext_start = ext_start;
last_ext_len = ext_len;

/* Copy this extent, looking for further opportunities to not
   bother to write zeros if --sparse=always, since SEEK_HOLE
   is conservative and may miss some holes. */
off_t n_read;
bool read_hole;
if (! sparse_copy (src_fd, dest_fd, abuf, buf_size,
                  sparse_mode != SPARSE_ALWAYS ? 0 : hole_size,
                  true, allow_relink, src_name, dst_name,
                  ext_len, &n_read, &read_hole))
    return false;

dest_pos = ext_start + n_read;
if (n_read)
    wrote_hole_at_eof = read_hole;
if (n_read < ext_len)
{
{
    /* The input file shrank. */
    src_total_size = dest_pos;
    break;
}

ext_start = lseek (src_fd, dest_pos, SEEK_DATA);
if (ext_start < 0 && errno != ENXIO)
    goto cannot_lseek;
}

/* When the source file ends with a hole, we have to do a little more work,
   since the above copied only up to and including the final extent.
   In order to complete the copy, we may have to insert a hole or write
   zeros in the destination corresponding to the source file's hole-at-EOF.

   In addition, if the final extent was a block of zeros at EOF and we've
   just converted them to a hole in the destination, we must call ftruncate
   here in order to record the proper length in the destination. */
if ((dest_pos < src_total_size || wrote_hole_at_eof)
    && ! (sparse_mode == SPARSE_NEVER
          ? write_zeros (dest_fd, src_total_size - dest_pos)
          : ftruncate (dest_fd, src_total_size) == 0))
{
{
    error (0, errno, _("failed to extend %s"), quoteaf (dst_name));
    return false;
}

if (sparse_mode == SPARSE_ALWAYS && dest_pos < src_total_size

```

```

    && punch_hole (dest_fd, dest_pos, src_total_size - dest_pos) < 0)
{
    error (0, errno, _("error deallocating %s"), quoteaf (dst_name));
    return false;
}

return true;

cannot_lseek:
error (0, errno, _("cannot lseek %s"), quoteaf (src_name));
return false;
}
#endif

/* FIXME: describe */
/* FIXME: rewrite this to use a hash table so we avoid the quadratic
   performance hit that's probably noticeable only on trees deeper
   than a few hundred levels. See use of active_dir_map in remove.c */

ATTRIBUTE_PURE
static bool
is_ancestor (const struct stat *sb, const struct dir_list *ancestors)
{
while (ancestors != 0)
{
    if (ancestors->ino == sb->st_ino && ancestors->dev == sb->st_dev)
        return true;
    ancestors = ancestors->parent;
}
return false;
}

static bool
errno_unsupported (int err)
{
return err == ENOTSUP || err == ENODATA;
}

#if USE_XATTR
ATTRIBUTE_FORMAT ((printf, 2, 3))
static void
copy_attr_error (MAYBE_UNUSED struct error_context *ctx,
                 char const *fmt, ...)
{
if (!errno_unsupported (errno))
{
    int err = errno;
    va_list ap;

    /* use verror module to print error message */
    va_start (ap, fmt);
    verror (0, err, fmt, ap);
    va_end (ap);
}
}

ATTRIBUTE_FORMAT ((printf, 2, 3))
static void
copy_attr_allerror (MAYBE_UNUSED struct error_context *ctx,
                    char const *fmt, ...)

```

```

{
int err = errno;
va_list ap;

/* use perror module to print error message */
va_start (ap, fmt);
perror (0, err, fmt, ap);
va_end (ap);
}

static char const *
copy_attr_quote (MAYBE_UNUSED struct error_context *ctx, char const *str)
{
return quoteaf (str);
}

static void
copy_attr_free (MAYBE_UNUSED struct error_context *ctx,
               MAYBE_UNUSED char const *str)
{
}

/* Exclude SELinux extended attributes that are otherwise handled,
and are problematic to copy again. Also honor attributes
configured for exclusion in /etc/xattr.conf.
FIXME: Should we handle POSIX ACLs similarly?
Return zero to skip. */
static int
check_selinux_attr (char const *name, struct error_context *ctx)
{
return STRNCMP_LIT (name, "security.selinux")
    && attr_copy_check_permissions (name, ctx);
}

/* If positive SRC_FD and DST_FD descriptors are passed,
then copy by fd, otherwise copy by name. */

static bool
copy_attr (char const *src_path, int src_fd,
           char const *dst_path, int dst_fd, struct cp_options const *x)
{
bool all_errors = (!x->data_copy_required || x->require_preserve_xattr);
bool some_errors = (!all_errors && !x->reduce_diagnostics);
int (*check) (char const *, struct error_context *)
    = (x->preserve_security_context || x->set_security_context
       ? check_selinux_attr : nullptr);

# if 4 < __GNUC__ + (8 <= __GNUC_MINOR__)
/* Pacify gcc -Wsuggest-attribute=format through at least GCC 13.2.1. */
# pragma GCC diagnostic push
# pragma GCC diagnostic ignored "-Wsuggest-attribute=format"
# endif
struct error_context *ctx
    = (all_errors || some_errors
       ? (&(struct error_context) {
           .error = all_errors ? copy_attr_allerror : copy_attr_error,
           .quote = copy_attr_quote,
           .quote_free = copy_attr_free
         })
       : nullptr);

```

```

# if 4 < __GNUC__ + (8 <= __GNUC_MINOR__)
# pragma GCC diagnostic pop
# endif

return ! (0 <= src_fd && 0 <= dst_fd
    ? attr_copy_fd (src_path, src_fd, dst_path, dst_fd, check, ctx)
    : attr_copy_file (src_path, dst_path, check, ctx));
}
#endif /* USE_XATTR */

static bool
copy_attr (MAYBE_UNUSED char const *src_path,
    MAYBE_UNUSED int src_fd,
    MAYBE_UNUSED char const *dst_path,
    MAYBE_UNUSED int dst_fd,
    MAYBE_UNUSED struct cp_options const *x)
{
    return true;
}
#endif /* USE_XATTR */

/* Read the contents of the directory SRC_NAME_IN, and recursively
copy the contents to DST_NAME_IN aka DST_DIRFD+DST_RELNAME_IN.
NEW_DST is true if DST_NAME_IN is a directory
that was created previously in the recursion.
SRC_SB and ANCESTORS describe SRC_NAME_IN.
Set *COPY_INTO_SELF if SRC_NAME_IN is a parent of
(or the same as) DST_NAME_IN; otherwise, clear it.
Propagate *FIRST_DIR_CREATED_PER_COMMAND_LINE_ARG from
caller to each invocation of copy_internal. Be careful to
pass the address of a temporary, and to update
*FIRST_DIR_CREATED_PER_COMMAND_LINE_ARG only upon completion.
Return true if successful. */

static bool
copy_dir (char const *src_name_in, char const *dst_name_in,
    int dst_dirfd, char const *dst_relname_in, bool new_dst,
    const struct stat *src_sb, struct dir_list *ancestors,
    const struct cp_options *x,
    bool *first_dir_created_per_command_line_arg,
    bool *copy_into_self)
{
    char *name_space;
    char *namep;
    struct cp_options non_command_line_options = *x;
    bool ok = true;

    name_space = savedir (src_name_in, SAVEDIR_SORT_FASTREAD);
    if (name_space == nullptr)
    {
        /* This diagnostic is a bit vague because savedir can fail in
           several different ways. */
        error (0, errno, _("cannot access %s"), quoteaf (src_name_in));
        return false;
    }

    /* For cp's -H option, dereference command line arguments, but do not
       dereference symlinks that are found via recursive traversal. */
    if (x->dereference == DEREF_COMMAND_LINE_ARGUMENTS)
        non_command_line_options.dereference = DEREF_NEVER;

```

```

bool new_first_dir_created = false;
namep = name_space;
while (*namep != '\0')
{
    bool local_copy_into_self;
    char *src_name = file_name_concat (src_name_in, namep, nullptr);
    char *dst_name = file_name_concat (dst_name_in, namep, nullptr);
    bool first_dir_created = *first_dir_created_per_command_line_arg;
    bool rename_succeeded;

    ok &= copy_internal (src_name, dst_name, dst_dirfd,
                         dst_name + (dst_relname_in - dst_name_in),
                         new_dst, src_sb,
                         ancestors, &non_command_line_options, false,
                         &first_dir_created,
                         &local_copy_into_self, &rename_succeeded);
    *copy_into_self |= local_copy_into_self;

    free (dst_name);
    free (src_name);

    /* If we're copying into self, there's no point in continuing,
       and in fact, that would even infloop, now that we record only
       the first created directory per command line argument. */
    if (local_copy_into_self)
        break;

    new_first_dir_created |= first_dir_created;
    namep += strlen (namep) + 1;
}
free (name_space);
*first_dir_created_per_command_line_arg = new_first_dir_created;

return ok;
}

/* Change the file mode bits of the file identified by DESC or
DIRFD+NAME to MODE. Use DESC if DESC is valid and fchmod is
available, DIRFD+NAME otherwise. */

static int
fchmod_or_lchmod (int desc, int dirfd, char const *name, mode_t mode)
{
#if HAVE_FCHMOD
if (0 <= desc)
    return fchmod (desc, mode);
#endif
return lchmodat (dirfd, name, mode);
}

/* Change the ownership of the file identified by DESC or
DIRFD+NAME to UID+GID. Use DESC if DESC is valid and fchown is
available, DIRFD+NAME otherwise. */

static int
fchown_or_lchown (int desc, int dirfd, char const *name, uid_t uid, gid_t gid)
{
#if HAVE_FCHOWN
if (0 <= desc)

```

```

        return fchown (desc, uid, gid);
#endif
return lchownat (dirfd, name, uid, gid);
}

/* Set the owner and owning group of DEST_DESC to the st_uid and
st_gid fields of SRC_SB. If DEST_DESC is undefined (-1), set
the owner and owning group of DST_NAME aka DST_DIRFD+DST_RELNAME
instead; for safety prefer lchownat since no
symbolic links should be involved. DEST_DESC must
refer to the same file as DST_NAME if defined.
Upon failure to set both UID and GID, try to set only the GID.
NEW_DST is true if the file was newly created; otherwise,
DST_SB is the status of the destination.
Return 1 if the initial syscall succeeds, 0 if it fails but it's OK
not to preserve ownership, -1 otherwise. */

static int
set_owner (const struct cp_options *x, char const *dst_name,
           int dst_dirfd, char const *dst_relname, int dest_desc,
           struct stat const *src_sb, bool new_dst,
           struct stat const *dst_sb)
{
uid_t uid = src_sb->st_uid;
gid_t gid = src_sb->st_gid;

/* Naively changing the ownership of an already-existing file before
changing its permissions would create a window of vulnerability if
the file's old permissions are too generous for the new owner and
group. Avoid the window by first changing to a restrictive
temporary mode if necessary. */

if (!new_dst && (x->preserve_mode || x->move_mode || x->set_mode))
{
mode_t old_mode = dst_sb->st_mode;
mode_t new_mode =
    (x->preserve_mode || x->move_mode ? src_sb->st_mode : x->mode);
mode_t restrictive_temp_mode = old_mode & new_mode & S_IRWXU;

if ((USE_ACL
     || (old_mode & CHMOD_MODE_BITS
         & (~new_mode | S_ISUID | S_ISGID | S_ISVTX)))
     && qset_acl (dst_name, dest_desc, restrictive_temp_mode) != 0)
{
if (!owner_failure_ok (x))
    error (0, errno, _("clearing permissions for %s"),
           quoteaf (dst_name));
return -x->require_preserve;
}
}

if (fchown_or_lchown (dest_desc, dst_dirfd, dst_relname, uid, gid) == 0)
    return 1;

/* The ownership change failed. If the failure merely means we lack
privileges to change owner+group, try to change just the group
and ignore any failure of this. Otherwise, report an error. */
if (chown_failure_ok (x))
    ignore_value (fchown_or_lchown (dest_desc, dst_dirfd, dst_relname,
                                   -1, gid));

```

```

else
{
    error (0, errno, _("failed to preserve ownership for %s"),
           quoteaf (dst_name));
    if (x->require_preserve)
        return -1;
}

return 0;
}

/* Set the st_author field of DEST_DESC to the st_author field of
   SRC_SB. If DEST_DESC is undefined (-1), set the st_author field
   of DST_NAME instead. DEST_DESC must refer to the same file as
   DST_NAME if defined. */

static void
set_author (char const *dst_name, int dest_desc, const struct stat *src_sb)
{
#ifndef HAVE_STRUCT_STAT_ST_AUTHOR
/* FIXME: Modify the following code so that it does not
   follow symbolic links. */

/* Preserve the st_author field. */
file_t file = (dest_desc < 0
               ? file_name_lookup (dst_name, 0, 0)
               : getdport (dest_desc));
if (file == MACH_PORT_NULL)
    error (0, errno, _("failed to lookup file %s"), quoteaf (dst_name));
else
{
    error_t err = file_chauthor (file, src_sb->st_author);
    if (err)
        error (0, err, _("failed to preserve authorship for %s"),
               quoteaf (dst_name));
    mach_port_deallocate (mach_task_self (), file);
}
#endif
(void) dst_name;
(void) dest_desc;
(void) src_sb;
#endif
}

/* Set the default security context for the process. New files will
have this security context set. Also existing files can have their
context adjusted based on this process context, by
set_file_security_ctx() called with PROCESS_LOCAL=true.
This should be called before files are created so there is no race
where a file may be present without an appropriate security context.
Based on CP_OPTIONS, diagnose warnings and fail when appropriate.
Return FALSE on failure, TRUE on success. */

bool
set_process_security_ctx (char const *src_name, char const *dst_name,
                        mode_t mode, bool new_dst, const struct cp_options *x)
{
if (x->preserve_security_context)
{
    /* Set the default context for the process to match the source. */

```

```

bool all_errors = !x->data_copy_required || x->require_preserve_context;
bool some_errors = !all_errors && !x->reduce_diagnostics;
char *con_raw;

if (0 <= lgetfilecon_raw (src_name, &con_raw))
{
    if (setfscreatecon_raw (con_raw) < 0)
    {
        if (all_errors || (some_errors && !errno_unsupported (errno)))
            error (0, errno,
                   _("failed to set default file creation context to %s"),
                   quote (con_raw));
        if (x->require_preserve_context)
        {
            freecon (con_raw);
            return false;
        }
    }
    freecon (con_raw);
}
else
{
    if (all_errors || (some_errors && !errno_unsupported (errno)))
    {
        error (0, errno,
               _("failed to get security context of %s"),
               quoteaf (src_name));
    }
    if (x->require_preserve_context)
        return false;
}
else if (x->set_security_context)
{
    /* With -Z, adjust the default context for the process
       to have the type component adjusted as per the destination path. */
    if (new_dst && defaultcon (x->set_security_context, dst_name, mode) < 0
        && ! ignorable_ctx_err (errno))
    {
        error (0, errno,
               _("failed to set default file creation context for %s"),
               quoteaf (dst_name));
    }
}

return true;
}

/* Reset the security context of DST_NAME, to that already set
   as the process default if !X->set_security_context. Otherwise
   adjust the type component of DST_NAME's security context as
   per the system default for that path. Issue warnings upon
   failure, when allowed by various settings in X.
   Return false on failure, true on success. */

bool
set_file_security_ctx (char const *dst_name,
                      bool recurse, const struct cp_options *x)
{
bool all_errors = (!x->data_copy_required

```

```

    || x->require_preserve_context);
bool some_errors = !all_errors && !x->reduce_diagnostics;

if (! restorecon (x->set_security_context, dst_name, recurse))
{
    if (all_errors || (some_errors && !errno_unsupported (errno)))
        error (0, errno, _("failed to set the security context of %s"),
               quoteaf_n (0, dst_name));
    return false;
}

return true;
}

#ifndef HAVE_STRUCT_STAT_ST_BLOCKS
#define HAVE_STRUCT_STAT_ST_BLOCKS 0
#endif

/* Type of scan being done on the input when looking for sparseness. */
enum scantype
{
/* An error was found when determining scantype. */
ERROR_SCANTYPE,
/* No fancy scanning; just read and write. */
PLAIN_SCANTYPE,
/* Read and examine data looking for zero blocks; useful when
   attempting to create sparse output. */
ZERO_SCANTYPE,
/* lseek information is available. */
LSEEK_SCANTYPE,
};

/* Result of infer_scantype. */
union scan_inference
{
/* Used if infer_scantype returns LSEEK_SCANTYPE. This is the
   offset of the first data block, or -1 if the file has no data. */
off_t ext_start;
};

/* Return how to scan a file with descriptor FD and stat buffer SB.
*SCAN_INFERENCE is set to a valid value if returning LSEEK_SCANTYPE. */
static enum scantype
infer_scantype (int fd, struct stat const *sb,
                union scan_inference *scan_inference)
{
scan_inference->ext_start = -1; /* avoid -Wmaybe-uninitialized */

/* Only attempt SEEK_HOLE if this heuristic
   suggests the file is sparse. */
if (! (HAVE_STRUCT_STAT_ST_BLOCKS
      && S_ISREG (sb->st_mode)
      && STP_NBLOCKS (sb) < sb->st_size / ST_NBLOCKSIZE))
    return PLAIN_SCANTYPE;

#ifndef SEEK_HOLE
off_t ext_start = lseek (fd, 0, SEEK_DATA);

```

```

if (0 <= ext_start || errno == ENXIO)
{
    scan_inference->ext_start = ext_start;
    return LSEEK_SCANTYPE;
}
else if (errno != EINVAL && !is_ENOTSUP (errno))
    return ERROR_SCANTYPE;
#endif

return ZERO_SCANTYPE;
}

#if HAVE_FCLONEFILEAT && !USE_XATTR
# include <sys/acl.h>
/* Return true if FD has a nontrivial ACL. */
static bool
fd_has_acl (int fd)
{
/* Every platform with fclonefileat (macOS 10.12 or later) also has
   acl_get_fd_np. */
bool has_acl = false;
acl_t acl = acl_get_fd_np (fd, ACL_TYPE_EXTENDED);
if (acl)
{
    acl_entry_t ace;
    has_acl = 0 <= acl_get_entry (acl, ACL_FIRST_ENTRY, &ace);
    acl_free (acl);
}
return has_acl;
}
#endif

/* Handle failure from FICLONE or fclonefileat.
Return FALSE if it's a terminal failure for this file. */

static bool
handle_clone_fail (int dst_dirfd, char const *dst_relname,
                  char const *src_name, char const *dst_name,
                  int dest_desc, bool new_dst, enum Reflink_type reflink_mode)
{
/* When the clone operation fails, report failure only with errno values
   known to mean trouble when the clone is supported and called properly.
   Do not report failure merely because !is_CLONENOTSUP (errno),
   as systems may yield oddball errno values here with FICLONE,
   and is_CLONENOTSUP is not appropriate for fclonefileat. */
bool report_failure = is_terminal_error (errno);

if (reflink_mode == REFLINK_ALWAYS || report_failure)
    error (0, errno, _("failed to clone %s from %s"),
           quoteaf_n (0, dst_name), quoteaf_n (1, src_name));

/* Remove the destination if cp --reflink=always created it
   but cloned no data. */
if (new_dst /* currently not for fclonefileat(). */
    && reflink_mode == REFLINK_ALWAYS
    && (! report_failure) || lseek (dest_desc, 0, SEEK_END) == 0
    && unlinkat (dst_dirfd, dst_relname, 0) != 0 && errno != ENOENT)
    error (0, errno, _("cannot remove %s"), quoteaf (dst_name));

if (! report_failure)

```

```

copy_debug.reflink = COPY_DEBUG_UNSUPPORTED;

if (reflink_mode == REFLINK_ALWAYS || report_failure)
    return false;

return true;
}

/* Copy a regular file from SRC_NAME to DST_NAME aka DST_DIRFD+DST_RELNAME.
If the source file contains holes, copies holes and blocks of zeros
in the source file as holes in the destination file.
(Holes are read as zeroes by the 'read' system call.)
When creating the destination, use DST_MODE & ~OMITTED_PERMISSIONS
as the third argument in the call to open, adding
OMITTED_PERMISSIONS after copying as needed.
X provides many option settings.
Return true if successful.
*NEW_DST is initially as in copy_internal.
If successful, set *NEW_DST to true if the destination file was created and
to false otherwise; if unsuccessful, perhaps set *NEW_DST to some value.
SRC_SB is the result of calling follow_fstatat on SRC_NAME;
it might be updated by calling fstat again on the same file,
to give it slightly more up-to-date contents. */

static bool
copy_reg (char const *src_name, char const *dst_name,
          int dst_dirfd, char const *dst_relname,
          const struct cp_options *x,
          mode_t dst_mode, mode_t omitted_permissions, bool *new_dst,
          struct stat *src_sb)
{
    char *buf = nullptr;
    int dest_desc;
    int dest_errno;
    int source_desc;
    mode_t extra_permissions;
    struct stat sb;
    struct stat src_open_sb;
    union scan_inference scan_inference;
    bool return_val = true;
    bool data_copy_required = x->data_copy_required;
    bool preserve_xattr = USE_XATTR & x->preserve_xattr;

    copy_debug.offload = COPY_DEBUG_UNKNOWN;
    copy_debug.reflink = x->reflink_mode ? COPY_DEBUG_UNKNOWN : COPY_DEBUG_NO;
    copy_debug.sparse_detection = COPY_DEBUG_UNKNOWN;

    source_desc = open (src_name,
                       (O_RDONLY | O_BINARY
                        | (x->dereference == DEREF_NEVER ? O_NOFOLLOW : 0)));
    if (source_desc < 0)
    {
        error (0, errno, _("cannot open %s for reading"), quoteaf (src_name));
        return false;
    }

    if (fstat (source_desc, &src_open_sb) != 0)
    {
        error (0, errno, _("cannot fstat %s"), quoteaf (src_name));

```

```

return_val = false;
goto close_src_desc;
}

/* Compare the source dev/ino from the open file to the incoming,
   saved ones obtained via a previous call to stat. */
if (!psame_inode (src_sb, &src_open_sb))
{
    error (0, 0,
           _("skipping file %s, as it was replaced while being copied"),
           quoteaf (src_name));
    return_val = false;
    goto close_src_desc;
}

/* Might as well tell the caller about the latest version of the
   source file status, since we have it already. */
*src_sb = src_open_sb;
mode_t src_mode = src_sb->st_mode;

/* The semantics of the following open calls are mandated
   by the specs for both cp and mv. */
if (!*new_dst)
{
    int open_flags =
        O_WRONLY | O_BINARY | (data_copy_required ? O_TRUNC : 0);
    dest_desc = openat (dst_dirfd, dst_relnname, open_flags);
    dest_errno = errno;

    /* When using cp --preserve=context to copy to an existing destination,
       reset the context as per the default context, which has already been
       set according to the src.
       When using the mutually exclusive -Z option, then adjust the type of
       the existing context according to the system default for the dest.
       Note we set the context here, _after_ the file is opened, lest the
       new context disallow that. */
    if (0 <= dest_desc
        && (x->set_security_context || x->preserve_security_context))
    {
        if (!set_file_security_ctx (dst_name, false, x))
        {
            if (x->require_preserve_context)
            {
                return_val = false;
                goto close_src_and_dst_desc;
            }
        }
    }

    if (dest_desc < 0 && dest_errno != ENOENT
        && x->unlink_dest_after_failed_open)
    {
        if (unlinkat (dst_dirfd, dst_relnname, 0) == 0)
        {
            if (x->verbose)
                printf (_("removed %s\n"), quoteaf (dst_name));
        }
        else if (errno != ENOENT)
        {
            error (0, errno, _("cannot remove %s"), quoteaf (dst_name));
        }
    }
}

```

```

        return_val = false;
        goto close_src_desc;
    }

    dest_errno = ENOENT;
}

if (dest_desc < 0 && dest_errno == ENOENT)
{
    /* Ensure there is no race where a file may be left without
       an appropriate security context. */
    if (x->set_security_context)
    {
        if (! set_process_security_ctx (src_name, dst_name, dst_mode,
                                       true, x))
        {
            return_val = false;
            goto close_src_desc;
        }
    }

    /* Tell caller that the destination file is created. */
    *new_dst = true;
}
}

if (*new_dst)
{
#ifndef HAVE_FCLONEFILEAT && !USE_XATTR
#ifndef CLONE_ACL
#define CLONE_ACL 0 /* Added in macOS 12.6. */
#endif
#endif
#ifndef CLONE_NOOWNERCOPY
#define CLONE_NOOWNERCOPY 0 /* Added in macOS 10.13. */
#endif
#endif
/* Try fclonefileat if copying data in reflink mode.
   Use CLONE_NOFOLLOW to avoid security issues that could occur
   if writing through dangling symlinks. Although the circa
   2023 macOS documentation doesn't say so, CLONE_NOFOLLOW
   affects the destination file too. */
if (data_copy_required && x->reflink_mode
    && (CLONE_NOOWNERCOPY || x->preserve_ownership))
{
    /* Try fclonefileat so long as it won't create the
       destination with unwanted permissions, which could lead
       to a security race. */
    mode_t cloned_mode_bits = S_ISVTX | S_IRWXUGO;
    mode_t cloned_mode = src_mode & cloned_mode_bits;
    mode_t desired_mode
        = (x->preserve_mode ? src_mode & CHMOD_MODE_BITS
          : x->set_mode ? x->mode
          : ((x->explicit_no_preserve_mode ? MODE_RW_UGO : dst_mode)
             & ~ cached_umask ()));
    if (!(cloned_mode & ~desired_mode))
    {
        int fc_flags
            = (CLONE_NOFOLLOW
              | (x->preserve_mode ? CLONE_ACL : 0)
              | (x->preserve_ownership ? 0 : CLONE_NOOWNERCOPY));
        int s = fclonefileat (source_desc, dst_dirfd, dst_relname,

```

```

                fc_flags);
if (s != 0 && (fc_flags & CLONE_ACL) && errno == EINVAL)
{
    fc_flags &= ~CLONE_ACL;
    s = fclonefileat (source_desc, dst_dirfd, dst_relname,
                      fc_flags);
}
if (s == 0)
{
    copy_debug.reflink = COPY_DEBUG_YES;

/* Update the clone's timestamps and permissions
   as needed. */

if (!x->preserve_timestamps)
{
    struct timespec timespec[2];
    timespec[0].tv_nsec = timespec[1].tv_nsec = UTIME_NOW;
    if (utimensat (dst_dirfd, dst_relname, timespec,
                   AT_SYMLINK_NOFOLLOW)
        != 0)
    {
        error (0, errno, _("updating times for %s"),
               quoteaf (dst_name));
        return_val = false;
        goto close_src_desc;
    }
}

extra_permissions = desired_mode & ~cloned_mode;
if (!extra_permissions
    && (!x->preserve_mode || (fc_flags & CLONE_ACL)
         || !fd_has_acl (source_desc)))
{
    goto close_src_desc;
}

/* Either some desired permissions were not cloned,
   or ACLs were not cloned despite that being requested. */
omitted_permissions = 0;
dest_desc = -1;
goto set_dest_mode;
}
if (! handle_clone_fail (dst_dirfd, dst_relname, src_name,
                        dst_name,
                        -1, false /* We didn't create dst */,
                        x->reflink_mode))
{
    return_val = false;
    goto close_src_desc;
}
else
    copy_debug.reflink = COPY_DEBUG_AVOIDED;
}
else if (data_copy_required && x->reflink_mode)
{
    if (! CLONE_NOOWNERCOPY)
        copy_debug.reflink = COPY_DEBUG_AVOIDED;
}

```

```

#endif

/* To allow copying xattrs on read-only files, create with u+w.
   This satisfies an inode permission check done by
   xattr_permission in fs/xattr.c of the GNU/Linux kernel. */
mode_t open_mode =
  ((dst_mode & ~omitted_permissions)
   | (preserve_xattr && !x->owner_privileges ? S_IWUSR : 0));
extra_permissions = open_mode & ~dst_mode; /* either 0 or S_IWUSR */

int open_flags = O_WRONLY | O_CREAT | O_BINARY;
dest_desc = openat (dst_dirfd, dst_relname, open_flags | O_EXCL,
                    open_mode);
dest_errno = errno;

/* When trying to copy through a dangling destination symlink,
   the above open fails with EEXIST. If that happens, and
   readlinkat shows that it is a symlink, then we
   have a problem: trying to resolve this dangling symlink to
   a directory/destination-entry pair is fundamentally racy,
   so punt. If x->open_dangling_dest_symlink is set (cp sets
   that when POSIXLY_CORRECT is set in the environment), simply
   call open again, but without O_EXCL (potentially dangerous).
   If not, fail with a diagnostic. These shenanigans are necessary
   only when copying, i.e., not in move_mode. */
if (dest_desc < 0 && dest_errno == EEXIST && !x->move_mode)
{
    {
    char dummy[1];
    if (0 <= readlinkat (dst_dirfd, dst_relname, dummy, sizeof dummy))
        {
        if (x->open_dangling_dest_symlink)
            {
            dest_desc = openat (dst_dirfd, dst_relname,
                                open_flags, open_mode);
            dest_errno = errno;
            }
        else
            {
            error (0, 0, _("not writing through dangling symlink %s"),
                   quoteaf (dst_name));
            return_val = false;
            goto close_src_desc;
            }
        }
    }
}

/* Improve quality of diagnostic when a nonexistent dst_name
   ends in a slash and open fails with errno == EISDIR. */
if (dest_desc < 0 && dest_errno == EISDIR
    && *dst_name && dst_name[strlen (dst_name) - 1] == '/')
    dest_errno = ENOTDIR;
}

else
{
    omitted_permissions = extra_permissions = 0;
}

if (dest_desc < 0)
{
    error (0, dest_errno, _("cannot create regular file %s"),

```

```

        quoteaf (dst_name));
return_val = false;
goto close_src_desc;
}

#endif __MVS__
/* Copy MVS file tags */
_setfdccsid(dest_desc, (src_sb->st_tag.ft_txtflag << 16) | src_sb->st_tag.ft_ccsid);
#endif

/* --attributes-only overrides --reflink. */
if (data_copy_required && x->reflink_mode)
{
    if (clone_file (dest_desc, source_desc) == 0)
    {
        data_copy_required = false;
        copy_debug.reflink = COPY_DEBUG_YES;
    }
else
{
    if (! handle_clone_fail (dst_dirfd, dst_relname, src_name, dst_name,
                           dest_desc, *new_dst, x->reflink_mode))
    {
        return_val = false;
        goto close_src_and_dst_desc;
    }
}
}

if (! (data_copy_required | x->preserve_ownership | extra_permissions))
    sb.st_mode = 0;
else if (fstat (dest_desc, &sb) != 0)
{
    error (0, errno, _("cannot fstat %s"), quoteaf (dst_name));
    return_val = false;
    goto close_src_and_dst_desc;
}

/* If extra permissions needed for copy_xattr didn't happen (e.g.,
   due to umask) chmod to add them temporarily; if that fails give
   up with extra permissions, letting copy_attr fail later. */
mode_t temporary_mode = sb.st_mode | extra_permissions;
if (temporary_mode != sb.st_mode
    && (fchmod_or_lchmod (dest_desc, dst_dirfd, dst_relname, temporary_mode)
         != 0))
    extra_permissions = 0;

if (data_copy_required)
{
    /* Choose a suitable buffer size; it may be adjusted later. */
    size_t buf_size = io_blksize (&sb);
    size_t hole_size = STP_BLKSIZE (&sb);

    /* Deal with sparse files. */
    enum scantype scantype = infer_scantype (source_desc, &src_open_sb,
                                              &scan_inference);
    if (scantype == ERROR_SCANTYPE)
    {
        error (0, errno, _("cannot lseek %s"), quoteaf (src_name));
        return_val = false;
    }
}

```

```

        goto close_src_and_dst_desc;
    }
bool make_holes
    = (S_ISREG (sb.st_mode)
    && (x->sparse_mode == SPARSE_ALWAYS
    || (x->sparse_mode == SPARSE_AUTO
        && scantype != PLAIN_SCANTYPE)));

fdadvise (source_desc, 0, 0, FADVISE_SEQUENTIAL);

/* If not making a sparse file, try to use a more-efficient
   buffer size. */
if (! make_holes)
{
    /* Compute the least common multiple of the input and output
       buffer sizes, adjusting for outlandish values.
       Note we read in multiples of the reported block size
       to support (unusual) devices that have this constraint. */
    size_t blcm_max = MIN (SIZE_MAX, SSIZE_MAX);
    size_t blcm = buffer_lcm (io_blksize (&src_open_sb), buf_size,
                           blcm_max);

    /* Do not bother with a buffer larger than the input file, plus one
       byte to make sure the file has not grown while reading it. */
    if (S_ISREG (src_open_sb.st_mode) && src_open_sb.st_size < buf_size)
        buf_size = src_open_sb.st_size + 1;

    /* However, stick with a block size that is a positive multiple of
       blcm, overriding the above adjustments. Watch out for
       overflow. */
    buf_size += blcm - 1;
    buf_size -= buf_size % blcm;
    if (buf_size == 0 || blcm_max < buf_size)
        buf_size = blcm;
}

off_t n_read;
bool wrote_hole_at_eof = false;
if (! (
#endif SEEK_HOLE
    scantype == LSEEK_SCANTYPE
    ? lseek_copy (source_desc, dest_desc, &buf, buf_size, hole_size,
                  scan_inference.ext_start, src_open_sb.st_size,
                  make_holes ? x->sparse_mode : SPARSE_NEVER,
                  x->reflink_mode != REFLINK_NEVER,
                  src_name, dst_name)
    :
#endif
    sparse_copy (source_desc, dest_desc, &buf, buf_size,
                 make_holes ? hole_size : 0,
                 x->sparse_mode == SPARSE_ALWAYS,
                 x->reflink_mode != REFLINK_NEVER,
                 src_name, dst_name, UINTMAX_MAX, &n_read,
                 &wrote_hole_at_eof)))
{
    return_val = false;
    goto close_src_and_dst_desc;
}
else if (wrote_hole_at_eof && ftruncate (dest_desc, n_read) < 0)
{

```

```

        error (0, errno, _("failed to extend %s"), quoteaf (dst_name));
        return_val = false;
        goto close_src_and_dst_desc;
    }
}

if (x->preserve_timestamps)
{
    struct timespec timespec[2];
    timespec[0] = get_stat_atime (src_sb);
    timespec[1] = get_stat_mtime (src_sb);

    if (fdutimensat (dest_desc, dst_dirfd, dst_relname, timespec, 0) != 0)
    {
        error (0, errno, _("preserving times for %s"), quoteaf (dst_name));
        if (x->require_preserve)
        {
            return_val = false;
            goto close_src_and_dst_desc;
        }
    }
}

/* Set ownership before xattrs as changing owners will
   clear capabilities. */
if (x->preserve_ownership && ! SAME_OWNER_AND_GROUP (*src_sb, sb))
{
    switch (set_owner (x, dst_name, dst_dirfd, dst_relname, dest_desc,
                      src_sb, *new_dst, &sb))
    {
        case -1:
            return_val = false;
            goto close_src_and_dst_desc;

        case 0:
            src_mode &= ~ (S_ISUID | S_ISGID | S_ISVTX);
            break;
    }
}

if (preserve_xattr)
{
    if (!copy_attr (src_name, source_desc, dst_name, dest_desc, x)
        && x->require_preserve_xattr)
        return_val = false;
}

set_author (dst_name, dest_desc, src_sb);

#if HAVE_FCLONEFILEAT && !USE_XATTR
set_dest_mode:
#endif
if (x->preserve_mode || x->move_mode)
{
    if (copy_acl (src_name, source_desc, dst_name, dest_desc, src_mode) != 0
        && x->require_preserve)
        return_val = false;
}
else if (x->set_mode)
{

```

```

        if (set_acl (dst_name, dest_desc, x->mode) != 0)
            return_val = false;
    }
else if (x->explicit_no_preserve_mode && *new_dst)
{
    if (set_acl (dst_name, dest_desc, MODE_RW_UGO & ~cached_umask ()) != 0)
        return_val = false;
}
else if (omitted_permissions | extra_permissions)
{
    omitted_permissions &= ~ cached_umask ();
    if ((omitted_permissions | extra_permissions)
        && (fchmod_or_lchmod (dest_desc, dst_dirfd, dst_relname,
                             dst_mode & ~ cached_umask ())
            != 0))
    {
        error (0, errno, _("preserving permissions for %s"),
               quoteaf (dst_name));
        if (x->require_preserve)
            return_val = false;
    }
}

if (dest_desc < 0)
    goto close_src_desc;

close_src_and_dst_desc:
if (close (dest_desc) < 0)
{
    error (0, errno, _("failed to close %s"), quoteaf (dst_name));
    return_val = false;
}
close_src_desc:
if (close (source_desc) < 0)
{
    error (0, errno, _("failed to close %s"), quoteaf (src_name));
    return_val = false;
}

/* Output debug info for data copying operations. */
if (x->debug)
    emit_debug (x);

alignfree (buf);
return return_val;
}

/* Return whether it's OK that two files are the "same" by some measure.
The first file is SRC_NAME and has status SRC_SB.
The second is DST_DIRFD+DST_RELNAME and has status DST_SB.
The copying options are X. The goal is to avoid
making the 'copy' operation remove both copies of the file
in that case, while still allowing the user to e.g., move or
copy a regular file onto a symlink that points to it.
Try to minimize the cost of this function in the common case.
Set *RETURN_NOW if we've determined that the caller has no more
work to do and should return successfully, right away. */

static bool
same_file_ok (char const *src_name, struct stat const *src_sb,

```

```

int dst_dirfd, char const *dst_relname, struct stat const *dst_sb,
const struct cp_options *x, bool *return_now)
{
const struct stat *src_sb_link;
const struct stat *dst_sb_link;
struct stat tmp_dst_sb;
struct stat tmp_src_sb;

bool same_link;
bool same = psame_inode (src_sb, dst_sb);

*return_now = false;

/* FIXME: this should (at the very least) be moved into the following
if-block. More likely, it should be removed, because it inhibits
making backups. But removing it will result in a change in behavior
that will probably have to be documented -- and tests will have to
be updated. */
if (same && x->hard_link)
{
*return_now = true;
return true;
}

if (x->dereference == DEREF_NEVER)
{
same_link = same;

/* If both the source and destination files are symlinks (and we'll
know this here IFF preserving symlinks), then it's usually ok
when they are distinct. */
if (S_ISLNK (src_sb->st_mode) && S_ISLNK (dst_sb->st_mode))
{
bool sn = same_nameat (AT_FDCWD, src_name, dst_dirfd, dst_relname);
if (!sn)
{
/* It's fine when we're making any type of backup. */
if (x->backup_type != no_backups)
return true;

/* Here we have two symlinks that are hard-linked together,
and we're not making backups. In this unusual case, simply
returning true would lead to mv calling "rename(A,B)",
which would do nothing and return 0. */
if (same_link)
{
*return_now = true;
return ! x->move_mode;
}
}

return ! sn;
}

src_sb_link = src_sb;
dst_sb_link = dst_sb;
}
else
{
if (!same)

```

```

    return true;

    if (fstatat (dst_dirfd, dst_relname, &tmp_dst_sb,
                  AT_SYMLINK_NOFOLLOW) != 0
        || lstat (src_name, &tmp_src_sb) != 0)
        return true;

    src_sb_link = &tmp_src_sb;
    dst_sb_link = &tmp_dst_sb;

    same_link = psame_inode (src_sb_link, dst_sb_link);

    /* If both are symlinks, then it's ok, but only if the destination
       will be unlinked before being opened. This is like the test
       above, but with the addition of the unlink_dest_before_opening
       conjunct because otherwise, with two symlinks to the same target,
       we'd end up truncating the source file. */
    if (S_ISLNK (src_sb_link->st_mode) && S_ISLNK (dst_sb_link->st_mode)
        && x->unlink_dest_before_opening)
        return true;
    }

/* The backup code ensures there's a copy, so it's usually ok to
   remove any destination file. One exception is when both
   source and destination are the same directory entry. In that
   case, moving the destination file aside (in making the backup)
   would also rename the source file and result in an error. */
if (x->backup_type != no_backups)
{
    if (!same_link)
    {
        /* In copy mode when dereferencing symlinks, if the source is a
           symlink and the dest is not, then backing up the destination
           (moving it aside) would make it a dangling symlink, and the
           subsequent attempt to open it in copy_reg would fail with
           a misleading diagnostic. Avoid that by returning zero in
           that case so the caller can make cp (or mv when it has to
           resort to reading the source file) fail now. */

        /* FIXME-note: even with the following kludge, we can still provoke
           the offending diagnostic. It's just a little harder to do :-)
           $ rm -f a b c; touch c; ln -s c b; ln -s b a; cp -b a b
           cp: cannot open 'a' for reading: No such file or directory
           That's misleading, since a subsequent 'ls' shows that 'a'
           is still there.
           One solution would be to open the source file *before* moving
           aside the destination, but that'd involve a big rewrite. */
        if (!x->move_mode
            && x->dereference != DEREF_NEVER
            && S_ISLNK (src_sb_link->st_mode)
            && !S_ISLNK (dst_sb_link->st_mode))
            return false;

        return true;
    }

/* FIXME: What about case insensitive file systems ? */
return ! same_nameat (AT_FDCWD, src_name, dst_dirfd, dst_relname);
}

```

```

#ifndef 0
/* FIXME: use or remove */

/* If we're making a backup, we'll detect the problem case in
   copy_reg because SRC_NAME will no longer exist. Allowing
   the test to be deferred lets cp do some useful things.
   But when creating hardlinks and SRC_NAME is a symlink
   but DST_RELNAME is not we must test anyway. */
if (x->hard_link
    || !S_ISLNK (src_sb_link->st_mode)
    || S_ISLNK (dst_sb_link->st_mode))
    return true;

if (x->dereference != DEREF_NEVER)
    return true;
#endif

if (x->move_mode || x->unlink_dest_before_opening)
{
    /* They may refer to the same file if we're in move mode and the
       target is a symlink. That is ok, since we remove any existing
       destination file before opening it -- via 'rename' if they're on
       the same file system, via unlinkat otherwise. */
    if (S_ISLNK (dst_sb_link->st_mode))
        return true;

    /* It's not ok if they're distinct hard links to the same file as
       this causes a race condition and we may lose data in this case. */
    if (same_link
        && 1 < dst_sb_link->st_nlink
        && ! same_nameat (AT_FDCWD, src_name, dst_dirfd, dst_relname))
        return ! x->move_mode;
}

/* If neither is a symlink, then it's ok as long as they aren't
   hard links to the same file. */
if (!S_ISLNK (src_sb_link->st_mode) && !S_ISLNK (dst_sb_link->st_mode))
{
    if (!psame_inode (src_sb_link, dst_sb_link))
        return true;

    /* If they are the same file, it's ok if we're making hard links. */
    if (x->hard_link)
    {
        *return_now = true;
        return true;
    }
}

/* At this point, it is normally an error (data loss) to move a symlink
   onto its referent, but in at least one narrow case, it is not:
   In move mode, when
   1) src is a symlink,
   2) dest has a link count of 2 or more and
   3) dest and the referent of src are not the same directory entry,
      then it's ok, since while we'll lose one of those hard links,
      src will still point to a remaining link.
   Note that technically, condition #3 obviates condition #2, but we
   retain the 1 < st_nlink condition because that means fewer invocations
   of the more expensive #3.

```

```

Given this,
$ touch f && ln f l && ln -s f s
$ ls -og f l s
-rw-----. 2 0 Jan 4 22:46 f
-rw-----. 2 0 Jan 4 22:46 l
lrwxrwxrwx. 1 1 Jan 4 22:46 s -> f
this must fail: mv s f
this must succeed: mv s l */

if (x->move_mode
    && S_ISLNK (src_sb->st_mode)
    && 1 < dst_sb_link->st_nlink)
{
    char *abs_src = canonicalize_file_name (src_name);
    if (abs_src)
    {
        bool result = ! same_nameat (AT_FDCWD, abs_src,
                                     dst_dirfd, dst_relname);
        free (abs_src);
        return result;
    }
}

/* It's ok to recreate a destination symlink. */
if (x->symbolic_link && S_ISLNK (dst_sb_link->st_mode))
    return true;

if (x->dereference == DEREF_NEVER)
{
    if (! S_ISLNK (src_sb_link->st_mode))
        tmp_src_sb = *src_sb_link;
    else if (stat (src_name, &tmp_src_sb) != 0)
        return true;

    if (! S_ISLNK (dst_sb_link->st_mode))
        tmp_dst_sb = *dst_sb_link;
    else if (fstatat (dst_dirfd, dst_relname, &tmp_dst_sb, 0) != 0)
        return true;

    if (!psame_inode (&tmp_src_sb, &tmp_dst_sb))
        return true;

    if (x->hard_link)
    {
        /* It's ok to attempt to hardlink the same file,
           and return early if not replacing a symlink.
           Note we need to return early to avoid a later
           unlink() of DST (when SRC is a symlink). */
        *return_now = ! S_ISLNK (dst_sb_link->st_mode);
        return true;
    }
}

return false;
}

/* Return whether DST_DIRFD+DST_RELNAME, with mode MODE,
   is writable in the sense of 'mv'.
   Always consider a symbolic link to be writable. */
static bool

```

```

writable_destination (int dst_dirfd, char const *dst_relname, mode_t mode)
{
    return (S_ISLNK (mode)
        || can_write_any_file ()
        || faccessat (dst_dirfd, dst_relname, W_OK, AT_EACCESS) == 0);
}

static bool
overwrite_ok (struct cp_options const *x, char const *dst_name,
    int dst_dirfd, char const *dst_relname,
    struct stat const *dst_sb)
{
    if (! writable_destination (dst_dirfd, dst_relname, dst_sb->st_mode))
    {
        char perms[12];           /* "-rwxrwxrwx " ls-style modes. */
        strmode (dst_sb->st_mode, perms);
        perms[10] = '\0';
        fprintf (stderr,
            (x->move_mode || x->unlink_dest_before_opening
                || x->unlink_dest_after_failed_open)
            ? _("%s: replace %s, overriding mode %04lo (%s)? ")
            : _("%s: unwritable %s (mode %04lo, %s); try anyway? "),
            program_name, quoteaf (dst_name),
            (unsigned long int) (dst_sb->st_mode & CHMOD_MODE_BITS),
            &perms[1]);
    }
    else
    {
        fprintf (stderr, _("%s: overwrite %s? "),
            program_name, quoteaf (dst_name));
    }
    return yesno ();
}

/* Initialize the hash table implementing a set of F_triple entries
corresponding to destination files. */
extern void
dest_info_init (struct cp_options *x)
{
    x->dest_info
        = hash_initialize (DEST_INFO_INITIAL_CAPACITY,
            nullptr,
            triple_hash,
            triple_compare,
            triple_free);
    if (! x->dest_info)
        xalloc_die ();
}

/* Initialize the hash table implementing a set of F_triple entries
corresponding to source files listed on the command line. */
extern void
src_info_init (struct cp_options *x)
{
    /* Note that we use triple_hash_no_name here.
    Contrast with the use of triple_hash above.
    That is necessary because a source file may be specified
    in many different ways. We want to warn about this
}

```

```

cp a a d/
as well as this:
cp a ./a d/
*/
x->src_info
= hash_initialize (DEST_INFO_INITIAL_CAPACITY,
                  nullptr,
                  triple_hash_no_name,
                  triple_compare,
                  triple_free);
if (!x->src_info)
    xalloc_die ();
}

/* When effecting a move (e.g., for mv(1)), and given the name DST_NAME
aka DST_DIRFD+DST_RELNAME
of the destination and a corresponding stat buffer, DST_SB, return
true if the logical 'move' operation should _not_ proceed.
Otherwise, return false.
Depending on options specified in X, this code may issue an
interactive prompt asking whether it's ok to overwrite DST_NAME. */
static bool
abandon_move (const struct cp_options *x,
              char const *dst_name,
              int dst_dirfd, char const *dst_relname,
              struct stat const *dst_sb)
{
affirm (x->move_mode);
return (x->interactive == I_ALWAYS_NO
        || x->interactive == I_ALWAYS_SKIP
        || ((x->interactive == I_ASK_USER
              || (x->interactive == I_UNSPECIFIED
                  && x->stdin_tty
                  && !writable_destination (dst_dirfd, dst_relname,
                                             dst_sb->st_mode)))
            && !overwrite_ok (x, dst_name, dst_dirfd, dst_relname, dst_sb)));
}

/* Print --verbose output on standard output, e.g. 'new' -> 'old'.
If BACKUP_DST_NAME is non-null, then also indicate that it is
the name of a backup file. */
static void
emit_verbose (char const *format, char const *src, char const *dst,
              char const *backup_dst_name)
{
printf (format, quoteaf_n (0, src), quoteaf_n (1, dst));
if (backup_dst_name)
    printf (_(" (backup: %s)"), quoteaf (backup_dst_name));
putchar ('\n');
}

/* A wrapper around "setfscreatecon (nullptr)" that exits upon failure. */
static void
restore_default_fscreatecon_or_die (void)
{
if (setfscreatecon (nullptr) != 0)
    error (EXIT_FAILURE, errno,
           _("failed to restore the default file creation context"));
}

```

```

/* Return a newly-allocated string that is like STR
except replace its suffix SUFFIX with NEWSUFFIX. */
static char *
subst_suffix (char const *str, char const *suffix, char const *newsuffix)
{
idx_t prefixlen = suffix - str;
idx_t newsuffixsize = strlen (newsuffix) + 1;
char *r = ximalloc (prefixlen + newsuffixsize);
memcpy (r + prefixlen, newsuffix, newsuffixsize);
return memcpy (r, str, prefixlen);
}

/* Create a hard link to SRC_NAME aka SRC_DIRFD+SRC_RELNAME;
the new link is at DST_NAME aka DST_DIRFD+DST_RELNAME.
A null SRC_NAME stands for the file whose name is like DST_NAME
except with DST_RELNAME replaced with SRC_RELNAME.
Honor the REPLACE, VERBOSE and DEREference settings.
Return true upon success. Otherwise, diagnose the
failure and return false. If SRC_NAME is a symbolic link, then it will not
be followed unless DEREference is true.
If the system doesn't support hard links to symbolic links, then DST_NAME
will be created as a symbolic link to SRC_NAME. */
static bool
create_hard_link (char const *src_name, int src_dirfd, char const *src_relname,
                  char const *dst_name, int dst_dirfd, char const *dst_relname,
                  bool replace, bool verbose, bool dereference)
{
int err = force_linkat (src_dirfd, src_relname, dst_dirfd, dst_relname,
                        dereference ? AT_SYMLINK_FOLLOW : 0,
                        replace, -1);
if (0 < err)
{
    char *a_src_name = nullptr;
    if (!src_name)
        src_name = a_src_name = subst_suffix (dst_name, dst_relname,
                                              src_relname);
    error (0, err, _("cannot create hard link %s to %s"),
           quoteaf_n (0, dst_name), quoteaf_n (1, src_name));
    free (a_src_name);
    return false;
}
if (err < 0 && verbose)
    printf (_("removed %s\n"), quoteaf (dst_name));
return true;
}

/* Return true if the current file should be (tried to be) dereferenced:
either for DEREf_ALWAYS or for DEREf_COMMAND_LINE_ARGUMENTS in the case
where the current file is a COMMAND_LINE_ARG; otherwise return false. */
ATTRIBUTE_PURE
static inline bool
should_dereference (const struct cp_options *x, bool command_line_arg)
{
return x->dereference == DEREf_ALWAYS
    || (x->dereference == DEREf_COMMAND_LINE_ARGUMENTS
        && command_line_arg);
}

/* Return true if the source file with basename SRCPBASE and status SRC_ST

```

```

is likely to be the simple backup file for DST_DIRFD+DST_RELNAME. */
static bool
source_is_dst_backup (char const *srcbase, struct stat const *src_st,
                      int dst_dirfd, char const *dst_relname)
{
size_t srcbaselen = strlen (srcbase);
char const *dstbase = last_component (dst_relname);
size_t dstbaselen = strlen (dstbase);
size_t suffixlen = strlen (simple_backup_suffix);
if (! (srcbaselen == dstbaselen + suffixlen
      && memcmp (srcbase, dstbase, dstbaselen) == 0
      && STREQ (srcbase + dstbaselen, simple_backup_suffix)))
    return false;
char *dst_back = subst_suffix (dst_relname,
                               dst_relname + strlen (dst_relname),
                               simple_backup_suffix);
struct stat dst_back_sb;
int dst_back_status = fstatat (dst_dirfd, dst_back, &dst_back_sb, 0);
free (dst_back);
return dst_back_status == 0 && psame_inode (src_st, &dst_back_sb);
}

/* Copy the file SRC_NAME to the file DST_NAME aka DST_DIRFD+DST_RELNAME.
If NONEXISTENT_DST is positive, DST_NAME does not exist even as a
dangling symlink; if negative, it does not exist except possibly
as a dangling symlink; if zero, its existence status is unknown.
A non-null PARENT describes the parent directory.
ANCESTORS points to a linked, null terminated list of
devices and inodes of parent directories of SRC_NAME.
X summarizes the command-line options.
COMMAND_LINE_ARG means SRC_NAME was specified on the command line.
FIRST_DIR_CREATED_PER_COMMAND_LINE_ARG is both input and output.
Set *COPY_INTO_SELF if SRC_NAME is a parent of (or the
same as) DST_NAME; otherwise, clear it.
If X->move_mode, set *RENAME_SUCCEEDED according to whether
the source was simply renamed to the destination.
Return true if successful. */
static bool
copy_internal (char const *src_name, char const *dst_name,
               int dst_dirfd, char const *dst_relname,
               int nonexistent_dst,
               struct stat const *parent,
               struct dir_list *ancestors,
               const struct cp_options *x,
               bool command_line_arg,
               bool *first_dir_created_per_command_line_arg,
               bool *copy_into_self,
               bool *rename_succeeded)
{
struct stat src_sb;
struct stat dst_sb;
mode_t src_mode IF_LINT (= 0);
mode_t dst_mode IF_LINT (= 0);
mode_t dst_mode_bits;
mode_t omitted_permissions;
bool restore_dst_mode = false;
char *earlier_file = nullptr;
char *dst_backup = nullptr;
char const *drelname = *dst_relname ? dst_relname : ".";
bool delayed_ok;

```

```

bool copied_as_regular = false;
bool dest_is_symlink = false;
bool have_dst_lstat = false;

*copy_into_self = false;

int rename_errno = x->rename_errno;
if (x->move_mode && !x->exchange)
{
    if (rename_errno < 0)
        rename_errno = (renameatu (AT_FDCWD, src_name, dst_dirfd, drelname,
                               RENAME_NOREPLACE)
                      ? errno : 0);
    nonexistent_dst = *rename_succeeded = rename_errno == 0;
}

if (rename_errno == 0
    ? !x->last_file
    : rename_errno != EEXIST
    || (x->interactive != I_ALWAYS_NO && x->interactive != I_ALWAYS_SKIP))
{
    char const *name = rename_errno == 0 ? dst_name : src_name;
    int dirfd = rename_errno == 0 ? dst_dirfd : AT_FDCWD;
    char const *relname = rename_errno == 0 ? drelname : src_name;
    int fstatat_flags
        = x->dereference == DEREF_NEVER ? AT_SYMLINK_NOFOLLOW : 0;
    if (follow_fstatat (dirfd, relname, &src_sb, fstatat_flags) != 0)
    {
        error (0, errno, _("cannot stat %s"), quoteaf (name));
        return false;
    }
}

src_mode = src_sb.st_mode;

if (S_ISDIR (src_mode) && !x->recursive)
{
    error (0, 0, !x->install_mode /* cp */
          ? _("-r not specified; omitting directory %s")
          : _("omitting directory %s"),
          quoteaf (src_name));
    return false;
}
else
{
    #if defined lint && (defined __clang__ || defined __COVERITY__)
        affirm (x->move_mode);
        memset (&src_sb, 0, sizeof src_sb);
    #endif
}

/* Detect the case in which the same source file appears more than
   once on the command line and no backup option has been selected.
   If so, simply warn and don't copy it the second time.
   This check is enabled only if x->src_info is non-null. */
if (command_line_arg && x->src_info)
{
    if (!S_ISDIR (src_mode)
        && x->backup_type == no_backups
        && seen_file (x->src_info, src_name, &src_sb))

```

```

{
error (0, 0, _("warning: source file %s specified more than once"),
      quoteaf (src_name));
return true;
}

record_file (x->src_info, src_name, &src_sb);
}

bool dereference = should_dereference (x, command_line_arg);

/* Whether the destination is (or was) known to be new, updated as
more info comes in. This may become true if the destination is a
dangling symlink, in contexts where dangling symlinks should be
treated the same as nonexistent files. */
bool new_dst = 0 < nonexistent_dst;

if (! new_dst)
{
/* Normally, fill in DST_SB or set NEW_DST so that later code
can use DST_SB if NEW_DST is false. However, don't bother
doing this when rename_errno == EEXIST and X->interactive is
I_ALWAYS_NO or I_ALWAYS_SKIP, something that can happen only
with mv in which case x->update must be false which means
that even if !NEW_DST the move will be abandoned without
looking at DST_SB. */
if (! (rename_errno == EEXIST
      && (x->interactive == I_ALWAYS_NO
           || x->interactive == I_ALWAYS_SKIP)))
{
/* Regular files can be created by writing through symbolic
links, but other files cannot. So use stat on the
destination when copying a regular file, and lstat otherwise.
However, if we intend to unlink or remove the destination
first, use lstat, since a copy won't actually be made to the
destination in that case. */
bool use_lstat
= (! S_ISREG (src_mode)
  && (! x->copy_as_regular
        || (S_ISDIR (src_mode) && !x->keep_directory_symlink)
        || S_ISLNK (src_mode)))
  || x->move_mode || x->symbolic_link || x->hard_link
  || x->backup_type != no_backups
  || x->unlink_dest_before_opening);
if (!use_lstat && nonexistent_dst < 0)
    new_dst = true;
else if (0 <= follow_fstatat (dst_dirfd, drelname, &dst_sb,
                               use_lstat ? AT_SYMLINK_NOFOLLOW : 0))
{
have_dst_lstat = use_lstat;
rename_errno = EEXIST;
}
else if (errno == ENOENT)
    new_dst = true;
else if (errno == ELOOP && !use_lstat
      && x->unlink_dest_after_failed_open)
{
/* cp -f's destination might be a symlink loop.
   Leave new_dst=false so that we try to unlink later. */
}
}

```

```

else
{
    error (0, errno, _("cannot stat %s"), quoteaf (dst_name));
    return false;
}
}

if (rename_errno == EEXIST)
{
    bool return_now = false;
    bool return_val = true;
    bool skipped = false;

    if ((x->interactive != I_ALWAYS_NO && x->interactive != I_ALWAYS_SKIP)
        && ! same_file_ok (src_name, &src_sb, dst_dirfd, drelname,
                            &dst_sb, x, &return_now))
    {
        error (0, 0, ("%s and %s are the same file"),
               quoteaf_n (0, src_name), quoteaf_n (1, dst_name));
        return false;
    }

    if (x->update && !S_ISDIR (src_mode))
    {
        /* When preserving timestamps (but not moving within a file
           system), don't worry if the destination timestamp is
           less than the source merely because of timestamp
           truncation. */
        int options = ((x->preserve_timestamps
                       && ! (x->move_mode
                             && dst_sb.st_dev == src_sb.st_dev))
                      ? UTIMECMP_TRUNCATE_SOURCE
                      : 0);

        if (0 <= utimecmpat (dst_dirfd, dst_relname, &dst_sb,
                             &src_sb, options))
        {
            /* We're using --update and the destination is not older
               than the source, so do not copy or move. Pretend the
               rename succeeded, so the caller (if it's mv) doesn't
               end up removing the source file. */
            if (rename_succeeded)
                *rename_succeeded = true;

            /* However, we still must record that we've processed
               this src/dest pair, in case this source file is
               hard-linked to another one. In that case, we'll use
               the mapping information to link the corresponding
               destination names. */
            earlier_file = remember_copied (dst_relname, src_sb.st_ino,
                                            src_sb.st_dev);
            if (earlier_file)
            {
                /* Note we currently replace DST_NAME unconditionally,
                   even if it was a newer separate file. */
                if (! create_hard_link (nullptr, dst_dirfd, earlier_file,
                                      dst_name, dst_dirfd, dst_relname,
                                      true,
                                      x->verbose, dereference))
            }
        }
    }
}

```

```

        goto un_backup;
    }
}

skipped = true;
goto skip;
}
}

/* When there is an existing destination file, we may end up
   returning early, and hence not copying/moving the file.
   This may be due to an interactive 'negative' reply to the
   prompt about the existing file. It may also be due to the
   use of the --no-clobber option.

cp and mv treat -i and -f differently. */
if (x->move_mode)
{
    if (abandon_move (x, dst_name, dst_dirfd, drelname, &dst_sb))
    {
        /* Pretend the rename succeeded, so the caller (mv)
           doesn't end up removing the source file. */
        if (rename_succeeded)
            *rename_succeeded = true;

        skipped = true;
        return_val = x->interactive == I_ALWAYS_SKIP;
    }
}
else
{
    if (! S_ISDIR (src_mode)
        && (x->interactive == I_ALWAYS_NO
            || x->interactive == I_ALWAYS_SKIP
            || (x->interactive == I_ASK_USER
                && ! overwrite_ok (x, dst_name, dst_dirfd,
                                    dst_relname, &dst_sb))))
    {
        skipped = true;
        return_val = x->interactive == I_ALWAYS_SKIP;
    }
}

skip:
if (skipped)
{
    if (x->interactive == I_ALWAYS_NO)
        error (0, 0, _("not replacing %s"), quoteaf (dst_name));
    else if (x->debug)
        printf (_("skipped %s\n"), quoteaf (dst_name));

    return_now = true;
}

if (return_now)
    return return_val;

/* Copying a directory onto a non-directory, or vice versa,
   is ok only with --backup or --exchange. */
if (!S_ISDIR (src_mode) != !S_ISDIR (dst_sb.st_mode)
```

```

    && x->backup_type == no_backups && !x->exchange)
{
    error (0, 0,
        _S_ISDIR (src_mode)
        ? ("cannot overwrite non-directory %s "
            "with directory %s")
        : ("cannot overwrite directory %s "
            "with non-directory %s"),
        quoteaf_n (0, dst_name), quoteaf_n (1, src_name));
    return false;
}

/* Don't let the user destroy their data, even if they try hard:
   This mv command must fail (likewise for cp):
   rm -rf a b c; mkdir a b c; touch a/f b/f; mv a/f b/f c
   Otherwise, the contents of b/f would be lost.
   In the case of 'cp', b/f would be lost if the user simulated
   a move using cp and rm.
   Nothing is lost if you use --backup=numbered or --exchange. */
if (!S_ISDIR (dst_sb.st_mode) && command_line_arg
    && x->backup_type != numbered_backups && !x->exchange
    && seen_file (x->dest_info, dst_relname, &dst_sb))
{
    error (0, 0,
        _("will not overwrite just-created %s with %s"),
        quoteaf_n (0, dst_name), quoteaf_n (1, src_name));
    return false;
}

char const *srcbase;
if (x->backup_type != no_backups
    /* Don't try to back up a destination if the last
       component of src_name is "." or "..". */
    && !dot_or_dotdot (srcbase = last_component (src_name)))
    /* Create a backup of each destination directory in move mode,
       but not in copy mode. FIXME: it might make sense to add an
       option to suppress backup creation also for move mode.
       That would let one use mv to merge new content into an
       existing hierarchy. */
    && (x->move_mode || !S_ISDIR (dst_sb.st_mode)))
{
    /* Fail if creating the backup file would likely destroy
       the source file. Otherwise, the commands:
       cd /tmp; rm -f a a~; : > a; echo A > a~; cp --b=simple a~ a
       would leave two zero-length files: a and a~. */
    if (x->backup_type != numbered_backups
        && source_is_dst_backup (srcbase, &src_sb,
                                  dst_dirfd, dst_relname))
    {
        char const *fmt;
        fmt = (x->move_mode
            ? _("Backing up %s might destroy source; %s not moved")
            : _("Backing up %s might destroy source; %s not copied"));
        error (0, 0, fmt,
            quoteaf_n (0, dst_name),
            quoteaf_n (1, src_name));
        return false;
    }
}

char *tmp_backup = backup_file_rename (dst_dirfd, dst_relname,

```

```

        x->backup_type);

/* FIXME: use fts:
   Using alloca for a file name that may be arbitrarily
   long is not recommended. In fact, even forming such a name
   should be discouraged. Eventually, this code will be rewritten
   to use fts, so using alloca here will be less of a problem. */
if (tmp_backup)
{
    idx_t dirlen = dst_relname - dst_name;
    idx_t backupsize = strlen (tmp_backup) + 1;
    dst_backup = alloca (dirlen + backupsize);
    memcpy (mempcpy (dst_backup, dst_name, dirlen),
            tmp_backup, backupsize);
    free (tmp_backup);
}
else if (errno != ENOENT)
{
    error (0, errno, _("cannot backup %s"), quoteaf (dst_name));
    return false;
}
new_dst = true;
}
else if (! S_ISDIR (dst_sb.st_mode)
/* Never unlink dst_name when in move mode. */
&& ! x->move_mode
&& (x->unlink_dest_before_opening
|| (x->data_copy_required
&& ((x->preserve_links && 1 < dst_sb.st_nlink)
|| (x->dereference == DEREF_NEVER
&& ! S_ISREG (src_sb.st_mode))))
))
{
if (unlinkat (dst_dirfd, dst_relname, 0) != 0 && errno != ENOENT)
{
    error (0, errno, _("cannot remove %s"), quoteaf (dst_name));
    return false;
}
new_dst = true;
if (x->verbose)
    printf (_("removed %s\n"), quoteaf (dst_name));
}
}
}

/* Ensure we don't try to copy through a symlink that was
   created by a prior call to this function. */
if (command_line_arg
&& x->dest_info
&& ! x->move_mode
&& x->backup_type == no_backups)
{
/* If we did not follow symlinks above, good: use that data.
   Otherwise, use AT_SYMLINK_NOFOLLOW, in case dst_name is a symlink. */
struct stat tmp_buf;
struct stat *dst_lstat_sb
= (have_dst_lstat ? &dst_sb
:fstatat (dst_dirfd, drename, &tmp_buf, AT_SYMLINK_NOFOLLOW) < 0
? nullptr : &tmp_buf);

```

```

/* Never copy through a symlink we've just created. */
if (dst_lstat_sb
    && S_ISLNK (dst_lstat_sb->st_mode)
    && seen_file (x->dest_info, dst_relname, dst_lstat_sb))
{
    error (0, 0,
        _("will not copy %s through just-created symlink %s"),
        quoteaf_n (0, src_name), quoteaf_n (1, dst_name));
    return false;
}
}

/* If the source is a directory, we don't always create the destination
   directory. So --verbose should not announce anything until we're
   sure we'll create a directory. Also don't announce yet when moving
   so we can distinguish renames versus copies. */
if (x->verbose && !x->move_mode && IS_ISDIR (src_mode))
    emit_verbose ("%s -> %s", src_name, dst_name, dst_backup);

/* Associate the destination file name with the source device and inode
   so that if we encounter a matching dev/ino pair in the source tree
   we can arrange to create a hard link between the corresponding names
   in the destination tree.

```

When using the `--link (-l)` option, there is no need to take special measures, because (barring race conditions) files that are hard-linked in the source tree will also be hard-linked in the destination tree.

Sometimes, when preserving links, we have to record dev/ino even though `st_nlink == 1`:

- when in `move_mode`, since we may be moving a group of N hard-linked files (via two or more command line arguments) to a different partition; the links may be distributed among the command line arguments (possibly hierarchies) so that the link count of the final, once-linked source file is reduced to 1 when it is considered below. But in this case (for `mv`) we don't need to incur the expense of recording the dev/ino => name mapping; all we really need is a lookup, to see if the dev/ino pair has already been copied.
- when using `-H` and processing a command line argument; that command line argument could be a symlink pointing to another command line argument. With '`cp -H --preserve=link`', we hard-link those two destination files.
- likewise for `-L` except that it applies to all files, not just command line arguments.

Also, with `--recursive`, record dev/ino of each command-line directory. We'll use that info to detect this problem: `cp -R dir dir`. \*/

```

if (rename_errno == 0 || x->exchange)
    earlier_file = nullptr;
else if (x->recursive && S_ISDIR (src_mode))
{
    if (command_line_arg)
        earlier_file = remember_copied (dst_relname,
                                        src_sb.st_ino, src_sb.st_dev);
    else
        earlier_file = src_to_dest_lookup (src_sb.st_ino, src_sb.st_dev);
}
else if (x->move_mode && src_sb.st_nlink == 1)

```

```

{
    earlier_file = src_to_dest_lookup (src_sb.st_ino, src_sb.st_dev);
}
else if (x->preserve_links
    && !x->hard_link
    && (1 < src_sb.st_nlink
        || (command_line_arg
            && x->dereference == DEREF_COMMAND_LINE_ARGUMENTS)
        || x->dereference == DEREF_ALWAYS))
{
    earlier_file = remember_copied (dst_relname,
                                    src_sb.st_ino, src_sb.st_dev);
}

/* Did we copy this inode somewhere else (in this command line argument)
   and therefore this is a second hard link to the inode? */

if (earlier_file)
{
    /* Avoid damaging the destination file system by refusing to preserve
       hard-linked directories (which are found at least in Netapp snapshot
       directories). */
    if (S_ISDIR (src_mode))
    {
        /* If src_name and earlier_file refer to the same directory entry,
           then warn about copying a directory into itself. */
        if (same_nameat (AT_FDCWD, src_name, dst_dirfd, earlier_file))
        {
            error (0, 0, _("cannot copy a directory, %s, into itself, %s"),
                  quoteaf_n (0, top_level_src_name),
                  quoteaf_n (1, top_level_dst_name));
            *copy_into_self = true;
            goto un_backup;
        }
        else if (same_nameat (dst_dirfd, dst_relname,
                             dst_dirfd, earlier_file))
        {
            error (0, 0, _("warning: source directory %s "
                           "specified more than once"),
                  quoteaf (top_level_src_name));
            /* In move mode, if a previous rename succeeded, then
               we won't be in this path as the source is missing. If the
               rename previously failed, then that has been handled, so
               pretend this attempt succeeded so the source isn't removed. */
            if (x->move_mode && rename_succeeded)
                *rename_succeeded = true;
            /* We only do backups in move mode, and for non directories.
               So just ignore this repeated entry. */
            return true;
        }
        else if (x->dereference == DEREF_ALWAYS
            || (command_line_arg
                && x->dereference == DEREF_COMMAND_LINE_ARGUMENTS))
        {
            /* This happens when e.g., encountering a directory for the
               second or subsequent time via symlinks when cp is invoked
               with -R and -L. E.g.,
               rm -rf a b c d; mkdir a b c d; ln -s ..c a; ln -s ..c b;
               cp -RL a b d
            */
        }
    }
}

```

```

        }
    else
    {
        char *earlier = subst_suffix (dst_name, dst_relname,
                                     earlier_file);
        error (0, 0, _("will not create hard link %s to directory %s"),
               quoteaf_n (0, dst_name), quoteaf_n (1, earlier));
        free (earlier);
        goto un_backup;
    }
}
else
{
    if (! create_hard_link (nullptr, dst_dirfd, earlier_file,
                           dst_name, dst_dirfd, dst_relname,
                           true, x->verbose, dereference))
        goto un_backup;

    return true;
}
}

if (x->move_mode)
{
    if (rename_errno == EEXIST)
        rename_errno = ((renameat (AT_FDCWD, src_name, dst_dirfd, drename,
                                  x->exchange ? RENAME_EXCHANGE : 0)
                        == 0)
                       ? 0 : errno);

    if (rename_errno == 0)
    {
        if (x->verbose)
            emit_verbose (x->exchange
                          ? _("exchanged %s <-> %s")
                          : _("renamed %s -> %s"),
                          src_name, dst_name, dst_backup);

        if (x->set_security_context)
        {
            /* -Z failures are only warnings currently. */
            (void) set_file_security_ctx (dst_name, true, x);
        }
    }

    if (rename_succeeded)
        *rename_succeeded = true;
}

if (command_line_arg && !x->last_file)
{
    /* Record destination dev/ino/name, so that if we are asked
       to overwrite that file again, we can detect it and fail. */
    /* It's fine to use the _source_ stat buffer (src_sb) to get the
       _destination_ dev/ino, since the rename above can't have
       changed those, and 'mv' always uses lstat.
       We could limit it further by operating
       only on non-directories when !x->exchange. */
    record_file (x->dest_info, dst_relname, &src_sb);
}

return true;

```

```

}

/* FIXME: someday, consider what to do when moving a directory into
   itself but when source and destination are on different devices. */

/* This happens when attempting to rename a directory to a
   subdirectory of itself. */
if (rename_errno == EINVAL)
{
    /* FIXME: this is a little fragile in that it relies on rename(2)
       failing with a specific errno value. Expect problems on
       non-POSIX systems. */
    error (0, 0, _("cannot move %s to a subdirectory of itself, %s"),
           quoteaf_n (0, top_level_src_name),
           quoteaf_n (1, top_level_dst_name));

    /* Note that there is no need to call forget_created here,
       (compare with the other calls in this file) since the
       destination directory didn't exist before. */

    *copy_into_self = true;
    /* FIXME-cleanup: Don't return true here; adjust mv.c accordingly.
       The only caller that uses this code (mv.c) ends up setting its
       exit status to nonzero when copy_into_self is nonzero. */
    return true;
}

/* WARNING: there probably exist systems for which an inter-device
   rename fails with a value of errno not handled here.
   If/as those are reported, add them to the condition below.
   If this happens to you, please do the following and send the output
   to the bug-reporting address (e.g., in the output of cp --help):
   touch k; perl -e 'rename "k","/tmp/k" or print "$!,$!+0,\n"'
   where your current directory is on one partition and /tmp is the other.
   Also, please try to find the E* errno macro name corresponding to
   the diagnostic and parenthesized integer, and include that in your
   e-mail. One way to do that is to run a command like this
   find /usr/include/. -type f \
      | xargs grep 'define.*\<E[A-Z]*\>.*\<18\>' /dev/null
   where you'd replace '18' with the integer in parentheses that
   was output from the perl one-liner above.
   If necessary, of course, change '/tmp' to some other directory. */
if (rename_errno != EXDEV || x->no_copy || x->exchange)
{
    /* There are many ways this can happen due to a race condition.
       When something happens between the initial follow_fstatat and the
       subsequent rename, we can get many different types of errors.
       For example, if the destination is initially a non-directory
       or non-existent, but it is created as a directory, the rename
       fails. If two 'mv' commands try to rename the same file at
       about the same time, one will succeed and the other will fail.
       If the permissions on the directory containing the source or
       destination file are made too restrictive, the rename will
       fail. Etc. */
    char const *quoted_dst_name = quoteaf_n (1, dst_name);
    if (x->exchange)
        error (0, rename_errno, _("cannot exchange %s and %s"),
               quoteaf_n (0, src_name), quoted_dst_name);
    else
        switch (rename_errno)

```

```

{
    case EDQUOT: case EEXIST: case EISDIR: case EMLINK:
        case ENOSPC: case ETXTBSY:
#if ENOTEMPTY != EEXIST
        case ENOTEMPTY:
#endif
        /* The destination must be the problem. Don't mention
           the source as that is more likely to confuse the user
           than be helpful. */
        error (0, rename_errno, _("cannot overwrite %s"),
               quoted_dst_name);
        break;

    default:
        error (0, rename_errno, _("cannot move %s to %s"),
               quoteaf_n (0, src_name), quoted_dst_name);
        break;
}
forget_created (src_sb.st_ino, src_sb.st_dev);
return false;
}

/* The rename attempt has failed. Remove any existing destination
   file so that a cross-device 'mv' acts as if it were really using
   the rename syscall. Note both src and dst must both be directories
   or not, and this is enforced above. Therefore we check the src_mode
   and operate on dst_name here as a tighter constraint and also because
   src_mode is readily available here. */
if ((unlinkat (dst_dirfd, drelname,
               S_ISDIR (src_mode) ? AT_REMOVEDIR : 0)
!= 0)
&& errno != ENOENT)
{
    error (0, errno,
           _("inter-device move failed: %s to %s; unable to remove target"),
           quoteaf_n (0, src_name), quoteaf_n (1, dst_name));
    forget_created (src_sb.st_ino, src_sb.st_dev);
    return false;
}

if (x->verbose && !S_ISDIR (src_mode))
    emit_verbose (_("copied %s -> %s"), src_name, dst_name, dst_backup);
new_dst = true;
}

/* If the ownership might change, or if it is a directory (whose
   special mode bits may change after the directory is created),
   omit some permissions at first, so unauthorized users cannot nip
   in before the file is ready. */
dst_mode_bits = (x->set_mode ? x->mode : src_mode) & CHMOD_MODE_BITS;
omitted_permissions =
(dst_mode_bits
& (x->preserve_ownership ? S_IRWXG | S_IRWXO
   : S_ISDIR (src_mode) ? S_IWGRP | S_IWOTH
   : 0));
delayed_ok = true;

/* If required, set the default security context for new files.
   Also for existing files this is used as a reference

```

```

when copying the context with --preserve=context.
FIXME: Do we need to consider dst_mode_bits here? */
if (!set_process_security_ctx (src_name, dst_name, src_mode, new_dst, x))
    return false;

if (S_ISDIR (src_mode))
{
    struct dir_list *dir;

    /* If this directory has been copied before during the
       recursion, there is a symbolic link to an ancestor
       directory of the symbolic link. It is impossible to
       continue to copy this, unless we've got an infinite file system. */

    if (is_ancestor (&src_sb, ancestors))
    {
        error (0, 0, _("cannot copy cyclic symbolic link %s"),
               quoteaf (src_name));
        goto un_backup;
    }

    /* Insert the current directory in the list of parents. */

    dir = alloca (sizeof *dir);
    dir->parent = ancestors;
    dir->ino = src_sb.st_ino;
    dir->dev = src_sb.st_dev;

    if (new_dst || !S_ISDIR (dst_sb.st_mode))
    {
        /* POSIX says mkdir's behavior is implementation-defined when
           (src_mode & ~S_IRWXUGO) != 0. However, common practice is
           to ask mkdir to copy all the CHMOD_MODE_BITS, letting mkdir
           decide what to do with S_ISUID | S_ISGID | S_ISVTX. */
        mode_t mode = dst_mode_bits & ~omitted_permissions;
        if (mkdirat (dst_dirfd, drelname, mode) != 0)
        {
            error (0, errno, _("cannot create directory %s"),
                   quoteaf (dst_name));
            goto un_backup;
        }

        /* We need search and write permissions to the new directory
           for writing the directory's contents. Check if these
           permissions are there. */

        if (fstatat (dst_dirfd, drelname, &dst_sb, AT_SYMLINK_NOFOLLOW) != 0)
        {
            error (0, errno, _("cannot stat %s"), quoteaf (dst_name));
            goto un_backup;
        }
        else if ((dst_sb.st_mode & S_IRWXU) != S_IRWXU)
        {
            /* Make the new directory searchable and writable. */

            dst_mode = dst_sb.st_mode;
            restore_dst_mode = true;

            if (lchmodat (dst_dirfd, drelname, dst_mode | S_IRWXU) != 0)
            {

```

```

        error (0, errno, _("setting permissions for %s"),
               quoteaf (dst_name));
        goto un_backup;
    }
}

/* Record the created directory's inode and device numbers into
   the search structure, so that we can avoid copying it again.
   Do this only for the first directory that is created for each
   source command line argument. */
if (!*first_dir_created_per_command_line_arg)
{
    remember_copied (dst_relname, dst_sb.st_ino, dst_sb.st_dev);
    *first_dir_created_per_command_line_arg = true;
}

if (x->verbose)
{
    if (x->move_mode)
        printf (_("created directory %s\n"), quoteaf (dst_name));
    else
        emit_verbose ("%s -> %s", src_name, dst_name, nullptr);
}
else
{
    omitted_permissions = 0;
}

/* For directories, the process global context could be reset for
   descendants, so use it to set the context for existing dirs here.
   This will also give earlier indication of failure to set ctx. */
if (x->set_security_context || x->preserve_security_context)
    if (!set_file_security_ctx (dst_name, false, x))
    {
        if (x->require_preserve_context)
            goto un_backup;
    }
}

/* Decide whether to copy the contents of the directory. */
if (x->one_file_system && parent && parent->st_dev != src_sb.st_dev)
{
    /* Here, we are crossing a file system boundary and cp's -x option
       is in effect: so don't copy the contents of this directory. */
}
else
{
    /* Copy the contents of the directory. Don't just return if
       this fails -- otherwise, the failure to read a single file
       in a source directory would cause the containing destination
       directory not to have owner/perms set properly. */
    delayed_ok = copy_dir (src_name, dst_name, dst_dirfd, dst_relname,
                           new_dst, &src_sb, dir, x,
                           first_dir_created_per_command_line_arg,
                           copy_into_self);
}
else if (x->symbolic_link)
{
    dest_is_symlink = true;
}

```



```

                replace, false, dereference))
        goto un_backup;
    }
else if (S_ISREG (src_mode)
    || (x->copy_as_regular && !S_ISLNK (src_mode)))
{
copied_as_regular = true;
/* POSIX says the permission bits of the source file must be
   used as the 3rd argument in the open call. Historical
   practice passed all the source mode bits to 'open', but the extra
   bits were ignored, so it should be the same either way.

This call uses DST_MODE_BITS, not SRC_MODE. These are
normally the same, and the exception (where x->set_mode) is
used only by 'install', which POSIX does not specify and
where DST_MODE_BITS is what's wanted. */
if (! copy_reg (src_name, dst_name, dst_dirfd, dst_relname,
               x, dst_mode_bits & S_IRWXUGO,
               omitted_permissions, &new_dst, &src_sb))
    goto un_backup;
}
else if (S_ISFIFO (src_mode))
{
/* Use mknodat, rather than mkfifoat, because the former preserves
   the special mode bits of a fifo on Solaris 10, while mkfifoat
   does not. But fall back on mkfifoat, because on some BSD systems,
   mknodat always fails when asked to create a FIFO. */
mode_t mode = src_mode & ~omitted_permissions;
if (mknodat (dst_dirfd, dst_relname, mode, 0) != 0)
    if (mkfifoat (dst_dirfd, dst_relname, mode & ~S_IFIFO) != 0)
    {
        error (0, errno, _("cannot create fifo %s"), quoteaf (dst_name));
        goto un_backup;
    }
}
else if (S_ISBLK (src_mode) || S_ISCHR (src_mode) || S_ISSOCK (src_mode))
{
mode_t mode = src_mode & ~omitted_permissions;
if (mknodat (dst_dirfd, dst_relname, mode, src_sb.st_rdev) != 0)
{
    error (0, errno, _("cannot create special file %s"),
           quoteaf (dst_name));
    goto un_backup;
}
}
else if (S_ISLNK (src_mode))
{
char *src_link_val = areadlink_with_size (src_name, src_sb.st_size);
dest_is_symlink = true;
if (src_link_val == nullptr)
{
    error (0, errno, _("cannot read symbolic link %s"),
           quoteaf (src_name));
    goto un_backup;
}

int symlink_err = force_symlinkat (src_link_val, dst_dirfd, dst_relname,
                                   x->unlink_dest_after_failed_open, -1);
if (0 < symlink_err && x->update && !new_dst && S_ISLNK (dst_sb.st_mode)
    && dst_sb.st_size == strlen (src_link_val))

```

```

{
/* See if the destination is already the desired symlink.
FIXME: This behavior isn't documented, and seems wrong
in some cases, e.g., if the destination symlink has the
wrong ownership, permissions, or timestamps. */
char *dest_link_val =
    areadlinkat_with_size (dst_dirfd, dst_relname, dst_sb.st_size);
if (dest_link_val)
{
    if (STREQ (dest_link_val, src_link_val))
        symlink_err = 0;
    free (dest_link_val);
}
free (src_link_val);
if (0 < symlink_err)
{
    error (0, symlink_err, _("cannot create symbolic link %s"),
           quoteaf (dst_name));
    goto un_backup;
}

if (x->preserve_security_context)
    restore_default_fscreatecon_or_die ();

if (x->preserve_ownership)
{
    /* Preserve the owner and group of the just-'copied'
       symbolic link, if possible. */
    if (HAVE_LCHOWN
        && (lchownat (dst_dirfd, dst_relname,
                      src_sb.st_uid, src_sb.st_gid)
             != 0)
        && ! chown_failure_ok (x))
    {
        error (0, errno, _("failed to preserve ownership for %s"),
               dst_name);
        if (x->require_preserve)
            goto un_backup;
    }
    else
    {
        /* Can't preserve ownership of symlinks.
           FIXME: maybe give a warning or even error for symlinks
           in directories with the sticky bit set -- there, not
           preserving owner/group is a potential security problem. */
    }
}
else
{
    error (0, 0, ("%s has unknown file type"), quoteaf (src_name));
    goto un_backup;
}

/* With -Z or --preserve=context, set the context for existing files.
Note this is done already for copy_reg() for reasons described therein. */
if (!new_dst && !x->copy_as_regular && !S_ISDIR (src_mode)
    && (x->set_security_context || x->preserve_security_context))
{

```

```

if (!set_file_security_ctx (dst_name, false, x))
{
    if (x->require_preserve_context)
        goto un_backup;
}
}

if (command_line_arg && x->dest_info)
{
    /* Now that the destination file is very likely to exist,
       add its info to the set. */
    struct stat sb;
    if (fstatat (dst_dirfd, drename, &sb, AT_SYMLINK_NOFOLLOW) == 0)
        record_file (x->dest_info, dst_relname, &sb);
}

/* If we've just created a hard-link due to cp's --link option,
   we're done. */
if (x->hard_link && !S_ISDIR (src_mode)
    && !(CAN_HARDLINK_SYMLINKS && S_ISLNK (src_mode)
          && x->dereference == DEREF_NEVER))
    return delayed_ok;

if (copied_as_regular)
    return delayed_ok;

/* POSIX says that 'cp -p' must restore the following:
   - permission bits
   - setuid, setgid bits
   - owner and group
   If it fails to restore any of those, we may give a warning but
   the destination must not be removed.
   FIXME: implement the above. */

/* Adjust the times (and if possible, ownership) for the copy.
   chown turns off set[ug]id bits for non-root,
   so do the chmod last. */

if (x->preserve_timestamps)
{
    struct timespec timespec[2];
    timespec[0] = get_stat_atime (&src_sb);
    timespec[1] = get_stat_mtime (&src_sb);

    int utimensat_flags = dest_is_symlink ? AT_SYMLINK_NOFOLLOW : 0;
    if (utimensat (dst_dirfd, drename, timespec, utimensat_flags) != 0)
    {
        error (0, errno, _("preserving times for %s"), quoteaf (dst_name));
        if (x->require_preserve)
            return false;
    }
}

/* Avoid calling chown if we know it's not necessary. */
if (!dest_is_symlink && x->preserve_ownership
    && (new_dst || !SAME_OWNER_AND_GROUP (src_sb, dst_sb)))
{
    switch (set_owner (x, dst_name, dst_dirfd, drename, -1,
                      &src_sb, new_dst, &dst_sb))
    {

```

```

    case -1:
        return false;

    case 0:
        src_mode &= ~ (S_ISUID | S_ISGID | S_ISVTX);
        break;
    }

}

/* Set xattrs after ownership as changing owners will clear capabilities. */
if (x->preserve_xattr && ! copy_attr (src_name, -1, dst_name, -1, x)
    && x->require_preserve_xattr)
    return false;

/* The operations beyond this point may dereference a symlink. */
if (dest_is_symlink)
    return delayed_ok;

set_author (dst_name, -1, &src_sb);

if (x->preserve_mode || x->move_mode)
{
    if (copy_acl (src_name, -1, dst_name, -1, src_mode) != 0
        && x->require_preserve)
        return false;
}
else if (x->set_mode)
{
    if (set_acl (dst_name, -1, x->mode) != 0)
        return false;
}
else if (x->explicit_no_preserve_mode && new_dst)
{
    int default_permissions = S_ISDIR (src_mode) || S_ISSOCK (src_mode)
        ? S_IRWXUGO : MODE_RW_UGO;
    dst_mode = dst_sb.st_mode;
    if (S_ISDIR (src_mode)) /* Keep set-group-ID for directories. */
        default_permissions |= (dst_mode & S_ISGID);
    if (set_acl (dst_name, -1, default_permissions & ~cached_umask ()) != 0)
        return false;
}
else
{
    if (omitted_permissions)
    {
        omitted_permissions &= ~ cached_umask ();

        if (omitted_permissions && !restore_dst_mode)
        {
            /* Permissions were deliberately omitted when the file
               was created due to security concerns. See whether
               they need to be re-added now. It'd be faster to omit
               the lstat, but deducing the current destination mode
               is tricky in the presence of implementation-defined
               rules for special mode bits. */
            if (new_dst && (fstatat (dst_dirfd, drelname, &dst_sb,
                                         AT_SYMLINK_NOFOLLOW)
                           != 0))
            {
                error (0, errno, _("cannot stat %s"), quoteaf (dst_name));
            }
        }
    }
}

```

```

        return false;
    }
    dst_mode = dst_sb.st_mode;
    if (omitted_permissions & ~dst_mode)
        restore_dst_mode = true;
    }
}

if (restore_dst_mode)
{
    if (!chmodat (dst_dirfd, drelname, dst_mode | omitted_permissions)
        != 0)
    {
        error (0, errno, _("preserving permissions for %s"),
               quoteaf (dst_name));
        if (x->require_preserve)
            return false;
    }
}
}

return delayed_ok;

un_backup:

if (x->preserve_security_context)
    restore_default_fscreatecon_or_die ();

/* We have failed to create the destination file.
   If we've just added a dev/ino entry via the remember_copied
   call above (i.e., unless we've just failed to create a hard link),
   remove the entry associating the source dev/ino with the
   destination file name, so we don't try to 'preserve' a link
   to a file we didn't create. */
if (earlier_file == nullptr)
    forget_created (src_sb.st_ino, src_sb.st_dev);

if (dst_backup)
{
    char const *dst_relbackup = &dst_backup[dst_relname - dst_name];
    if (renameat (dst_dirfd, dst_relbackup, dst_dirfd, drename) != 0)
        error (0, errno, _("cannot un-backup %s"), quoteaf (dst_name));
    else
    {
        if (x->verbose)
            printf (_("%s -> %s (unbackup)\n"),
                    quoteaf_n (0, dst_backup), quoteaf_n (1, dst_name));
    }
}
return false;
}

static void
valid_options (const struct cp_options *co)
{
    affirm (VALID_BACKUP_TYPE (co->backup_type));
    affirm (VALID_SPARSE_MODE (co->sparse_mode));
    affirm (VALID_REFLINK_MODE (co->reflink_mode));
    affirm (!(co->hard_link && co->symbolic_link));
    affirm (!
}

```

```

        (co->reflink_mode == REFLINK_ALWAYS
         && co->sparse_mode != SPARSE_AUTO));
    }

/* Copy the file SRC_NAME to the file DST_NAME aka DST_DIRFD+DST_RELNAME.
If NONEXISTENT_DST is positive, DST_NAME does not exist even as a
dangling symlink; if negative, it does not exist except possibly
as a dangling symlink; if zero, its existence status is unknown.
OPTIONS summarizes the command-line options.
Set *COPY_INTO_SELF if SRC_NAME is a parent of (or the
same as) DST_NAME; otherwise, set clear it.
If X->move_mode, set *RENAME_SUCCEEDED according to whether
the source was simply renamed to the destination.
Return true if successful. */

extern bool
copy (char const *src_name, char const *dst_name,
      int dst_dirfd, char const *dst_relname,
      int nonexistent_dst, const struct cp_options *options,
      bool *copy_into_self, bool *rename_succeeded)
{
    valid_options (options);

/* Record the file names: they're used in case of error, when copying
   a directory into itself. I don't like to make these tools do *any*
   extra work in the common case when that work is solely to handle
   exceptional cases, but in this case, I don't see a way to derive the
   top level source and destination directory names where they're used.
   An alternative is to use COPY_INTO_SELF and print the diagnostic
   from every caller -- but I don't want to do that. */
    top_level_src_name = src_name;
    top_level_dst_name = dst_name;

    bool first_dir_created_per_command_line_arg = false;
    return copy_internal (src_name, dst_name, dst_dirfd, dst_relname,
                          nonexistent_dst, nullptr, nullptr,
                          options, true,
                          &first_dir_created_per_command_line_arg,
                          copy_into_self, rename_succeeded);
}

/* Set *X to the default options for a value of type struct cp_options. */

extern void
cp_options_default (struct cp_options *x)
{
    memset (x, 0, sizeof *x);
    #ifdef PRIV_FILE_CHOWN
    {
        priv_set_t *pset = priv_allocset ();
        if (!pset)
            xalloc_die ();
        if (getppriv (PRIV_EFFECTIVE, pset) == 0)
        {
            x->chown_privileges = priv_ismember (pset, PRIV_FILE_CHOWN);
            x->owner_privileges = priv_ismember (pset, PRIV_FILE_OWNER);
        }
        priv_freeset (pset);
    }
    #else

```

```

x->chown_privileges = x->owner_privileges = (geteuid () == ROOT_UID);
#endif
x->rename_errno = -1;
}

/* Return true if it's OK for chown to fail, where errno is
the error number that chown failed with and X is the copying
option set. */

extern bool
chown_failure_ok (struct cp_options const *x)
{
/* If non-root uses -p, it's ok if we can't preserve ownership.
But root probably wants to know, e.g. if NFS disallows it,
or if the target system doesn't support file ownership.

Treat EACCES like EPERM and EINVAL to work around a bug in Linux
CIFS <https://bugs.gnu.org/65599>. Although this means coreutils
will ignore EACCES errors that it should report, problems should
occur only when some other process is racing with coreutils and
coreutils is not immune to races anyway. */

return ((errno == EPERM || errno == EINVAL || errno == EACCES)
       && !x->chown_privileges);
}

/* Similarly, return true if it's OK for chmod and similar operations
to fail, where errno is the error number that chmod failed with and
X is the copying option set. */

static bool
owner_failure_ok (struct cp_options const *x)
{
return ((errno == EPERM || errno == EINVAL || errno == EACCES)
       && !x->owner_privileges);
}

/* Return the user's umask, caching the result.

FIXME: If the destination's parent directory has has a default ACL,
some operating systems (e.g., GNU/Linux's "POSIX" ACLs) use that
ACL's mask rather than the process umask. Currently, the callers
of cached_umask incorrectly assume that this situation cannot occur. */

extern mode_t
cached_umask (void)
{
static mode_t mask = (mode_t) -1;
if (mask == (mode_t) -1)
{
    mask = umask (0);
    umask (mask);
}
return mask;
}
"""

,
"patch": """
diff --git a/src/copy.c b/src/copy.c
index b9fff03..554e433 100644
--- a/src/copy.c

```

```

+++ b/src/copy.c
@@ -113,6 +113,10 @@
#define CAN_HARDLINK_SYMLINKS 0
#endif

+#ifdef __MVS__
+# include "zos-io.h"
+#endif
+
struct dir_list
{
    struct dir_list *parent;
@@ -1507,6 +1511,11 @@ copy_reg (char const *src_name, char const *dst_name,
    goto close_src_desc;
}

+#ifdef __MVS__
+/* Copy MVS file tags */
+__setfdccsid(dest_desc, (src_sb->st_tag.ft_txtflag << 16) | src_sb->st_tag.ft_ccsid);
+#endif
+
/* --attributes-only overrides --reflink. */
if (data_copy_required && x->reflink_mode)
{
    """
},
{
    "wrong_code": "
"""
/* cp.c -- file copying (main routines)
Copyright (C) 1989-2024 Free Software Foundation, Inc.
```

This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License  
along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Written by Torbjörn Granlund, David MacKenzie, and Jim Meyering. \*/

```
#include <config.h>
#include <stdio.h>
#include <sys/types.h>
#include <getopt.h>
#include <selinux/label.h>

#include "system.h"
#include "argmatch.h"
#include "assure.h"
#include "backupfile.h"
#include "copy.h"
#include "cp-hash.h"
```

```

#include "filenamecat.h"
#include "ignore-value.h"
#include "quote.h"
#include "stat-time.h"
#include "targetdir.h"
#include "utimens.h"
#include "acl.h"

/* The official name of this program (e.g., no 'g' prefix). */
#define PROGRAM_NAME "cp"

#define AUTHORS \
proper_name_lite ("Torbjorn Granlund", "Torbj\303\266rn Granlund"), \
proper_name ("David MacKenzie"), \
proper_name ("Jim Meyering")

/* Used by do_copy, make_dir_parents_private, and re_protect
to keep a list of leading directories whose protections
need to be fixed after copying.*/
struct dir_attr
{
    struct stat st;
    bool restore_mode;
    size_t slash_offset;
    struct dir_attr *next;
};

/* For long options that have no equivalent short option, use a
non-character as a pseudo short option, starting with CHAR_MAX + 1. */
enum
{
    ATTRIBUTES_ONLY_OPTION = CHAR_MAX + 1,
    COPY_CONTENTS_OPTION,
    DEBUG_OPTION,
    NO_PRESERVE_ATTRIBUTES_OPTION,
    PARENTS_OPTION,
    PRESERVE_ATTRIBUTES_OPTION,
    REFLINK_OPTION,
    SPARSE_OPTION,
    STRIP_TRAILING_SLASHES_OPTION,
    UNLINK_DEST_BEFORE_OPENING,
    KEEP_DIRECTORY_SYMLINK_OPTION
};

/* True if the kernel is SELinux enabled. */
static bool selinux_enabled;

/* If true, the command "cp x/e_file e_dir" uses "e_dir/x/e_file"
as its destination instead of the usual "e_dir/e_file." */
static bool parents_option = false;

/* Remove any trailing slashes from each SOURCE argument. */
static bool remove_trailing_slashes;

static char const *const sparse_type_string[] =
{
    "never", "auto", "always", nullptr
};
static enum Sparse_type const sparse_type[] =
{

```

```

SPARSE_NEVER, SPARSE_AUTO, SPARSE_ALWAYS
};

ARGMATCH_VERIFY (sparse_type_string, sparse_type);

static char const *const reflink_type_string[] =
{
"auto", "always", "never", nullptr
};
static enum Reflink_type const reflink_type[] =
{
REFLINK_AUTO, REFLINK_ALWAYS, REFLINK_NEVER
};
ARGMATCH_VERIFY (reflink_type_string, reflink_type);

static char const *const update_type_string[] =
{
"all", "none", "older", nullptr
};
static enum Update_type const update_type[] =
{
UPDATE_ALL, UPDATE_NONE, UPDATE_OLDER,
};
ARGMATCH_VERIFY (update_type_string, update_type);

static struct option const long_opts[] =
{
{"archive", no_argument, nullptr, 'a'},
 {"attributes-only", no_argument, nullptr, ATTRIBUTES_ONLY_OPTION},
 {"backup", optional_argument, nullptr, 'b'},
 {"copy-contents", no_argument, nullptr, COPY_CONTENTS_OPTION},
 {"debug", no_argument, nullptr, DEBUG_OPTION},
 {"dereference", no_argument, nullptr, 'L'},
 {"force", no_argument, nullptr, 'f'},
 {"interactive", no_argument, nullptr, 'i'},
 {"link", no_argument, nullptr, 'l'},
 {"no-clobber", no_argument, nullptr, 'n'}, /* Deprecated. */
 {"no-dereference", no_argument, nullptr, 'P'},
 {"no-preserve", required_argument, nullptr, NO_PRESERVE_ATTRIBUTES_OPTION},
 {"no-target-directory", no_argument, nullptr, 'T'},
 {"one-file-system", no_argument, nullptr, 'x'},
 {"parents", no_argument, nullptr, PARENTS_OPTION},
 {"path", no_argument, nullptr, PARENTS_OPTION}, /* Deprecated. */
 {"preserve", optional_argument, nullptr, PRESERVE_ATTRIBUTES_OPTION},
 {"recursive", no_argument, nullptr, 'R'},
 {"remove-destination", no_argument, nullptr, UNLINK_DEST_BEFORE_OPENING},
 {"sparse", required_argument, nullptr, SPARSE_OPTION},
 {"reflink", optional_argument, nullptr, REFLINK_OPTION},
 {"strip-trailing-slashes", no_argument, nullptr,
STRIP_TRAILING_SLASHES_OPTION},
 {"suffix", required_argument, nullptr, 'S'},
 {"symbolic-link", no_argument, nullptr, 's'},
 {"target-directory", required_argument, nullptr, 't'},
 {"update", optional_argument, nullptr, 'u'},
 {"verbose", no_argument, nullptr, 'v'},
 {"keep-directory-symlink", no_argument, nullptr,
KEEP_DIRECTORY_SYMLINK_OPTION},
 {GETOPT_SELINUX_CONTEXT_OPTION_DECL},
 {GETOPT_HELP_OPTION_DECL},
 {GETOPT_VERSION_OPTION_DECL},
 {nullptr, 0, nullptr, 0}
}

```

```

};

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    {
        printf (_("\
Usage: %s [OPTION]... [-T] SOURCE DEST\n\
or: %s [OPTION]... SOURCE... DIRECTORY\n\
or: %s [OPTION]... -t DIRECTORY SOURCE...\\n\
"),
                program_name, program_name, program_name);
        fputs (_("\
Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.\\n\
"),
                stdout);
    }

    emit_mandatory_arg_note ();

    fputs (_("\
-a, --archive           same as -dR --preserve=all\\n\
--attributes-only      don't copy the file data, just the attributes\\n\
--backup[=CONTROL]     make a backup of each existing destination file\\
\\n\
-b                   like --backup but does not accept an argument\\n\
--copy-contents        copy contents of special files when recursive\\n\
-d                   same as --no-dereference --preserve=links\\n\
"),
                stdout);
    fputs (_("\
--debug               explain how a file is copied. Implies -v\\n\\
"),
                stdout);
    fputs (_("\
-f, --force             if an existing destination file cannot be\\n\
                           opened, remove it and try again (this option\\n\
                           is ignored when the -n option is also used)\\n\
                           prompt before overwrite (overrides a previous -n\\
\\n\
                           option)\\n\\
-H                   follow command-line symbolic links in SOURCE\\n\
"),
                stdout);
    fputs (_("\
-l, --link              hard link files instead of copying\\n\
-L, --dereference       always follow symbolic links in SOURCE\\n\
"),
                stdout);
    fputs (_("\
-n, --no-clobber        (deprecated) silently skip existing files.\\n\
                           See also --update\\n\\
"),
                stdout);
    fputs (_("\
-P, --no-dereference    never follow symbolic links in SOURCE\\n\
"),
                stdout);
    fputs (_("\
-p                   same as --preserve=mode,ownership,timestamps\\n\
--preserve[=ATTR_LIST]  preserve the specified attributes\\n\
"),
                stdout);
    fputs (_("\
--no-preserve=ATTR_LIST don't preserve the specified attributes\\n\
--parents            use full source file name under DIRECTORY\\n\
"),
                stdout);
}
}

```

```

"), stdout);
    fputs (_("\
-R, -r, --recursive      copy directories recursively\n\
    --reflink[=WHEN]      control clone/CoW copies. See below\n\
    --remove-destination  remove each existing destination file before\n\
                           attempting to open it (contrast with --force)\n\
\n"), stdout);
    fputs (_("\
    --sparse=WHEN        control creation of sparse files. See below\n\
    --strip-trailing-slashes  remove any trailing slashes from each SOURCE\n\
                               argument\n\
"), stdout);
    fputs (_("\
-s, --symbolic-link    make symbolic links instead of copying\n\
-S, --suffix=SUFFIX    override the usual backup suffix\n\
-t, --target-directory=DIRECTORY  copy all SOURCE arguments into DIRECTORY\n\
-T, --no-target-directory  treat DEST as a normal file\n\
"), stdout);
    fputs (_("\
--update[=UPDATE]      control which existing files are updated;\n\
                           UPDATE={all,none,none-fail,older(default)}.\n\
-u                      equivalent to --update[=older]. See below\n\
"), stdout);
    fputs (_("\
-v, --verbose          explain what is being done\n\
"), stdout);
    fputs (_("\
    --keep-directory-symlink  follow existing symlinks to directories\n\
"), stdout);
    fputs (_("\
-x, --one-file-system   stay on this file system\n\
"), stdout);
    fputs (_("\
-Z                     set SELinux security context of destination\n\
                           file to default type\n\
    --context[=CTX]       like -Z, or if CTX is specified then set the\n\
                           SELinux or SMACK security context to CTX\n\
"), stdout);
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
    fputs (_("\
\n"))
ATTR_LIST is a comma-separated list of attributes. Attributes are 'mode' for\n\
permissions (including any ACL and xattr permissions), 'ownership' for user\n\
and group, 'timestamps' for file timestamps, 'links' for hard links, 'context'\n\
for security context, 'xattr' for extended attributes, and 'all' for all\n\
attributes.\n\
"), stdout);
    fputs (_("\
\n"))
By default, sparse SOURCE files are detected by a crude heuristic and the\n\
corresponding DEST file is made sparse as well. That is the behavior\n\
selected by --sparse=auto. Specify --sparse=always to create a sparse DEST\n\
file whenever the SOURCE file contains a long enough sequence of zero bytes.\n\
Use --sparse=never to inhibit creation of sparse files.\n\
"), stdout);
    emit_update_parameters_note ();
    fputs (_("\
\n"))
When --reflink[=always] is specified, perform a lightweight copy, where the\n\

```

```

data blocks are copied only when modified. If this is not possible the copy\n\
fails, or if --reflink=auto is specified, fall back to a standard copy.\n\
Use --reflink=never to ensure a standard copy is performed.\n\
"), stdout);
    emit_backup_suffix_note ();
    fputs (_("\
\n\
As a special case, cp makes a backup of SOURCE when the force and backup\n\
options are given and SOURCE and DEST are the same name for an existing,\n\
regular file.\n\
"), stdout);
    emit_ancillary_info (PROGRAM_NAME);
}
exit (status);
}

```

/\* Ensure that parents of CONST\_DST\_NAME have correct protections, for  
the --parents option. This is done after all copying has been  
completed, to allow permissions that don't include user write/execute.

DST\_SRC\_NAME is the suffix of CONST\_DST\_NAME that is the source file name,  
DST\_DIRFD+DST\_RELNAME is equivalent to CONST\_DST\_NAME, and  
DST\_RELNAME equals DST\_SRC\_NAME after skipping any leading '/'s.

ATTR\_LIST is a null-terminated linked list of structures that  
indicates the end of the filename of each intermediate directory  
in CONST\_DST\_NAME that may need to have its attributes changed.  
The command 'cp --parents --preserve a/b/c d/e\_dir' changes the  
attributes of the directories d/e\_dir/a and d/e\_dir/a/b to match  
the corresponding source directories regardless of whether they  
existed before the 'cp' command was given.

Return true if the parent of CONST\_DST\_NAME and any intermediate  
directories specified by ATTR\_LIST have the proper permissions  
when done. \*/

```

static bool
re_protect (char const *const_dst_name, char const *dst_src_name,
            int dst_dirfd, char const *dst_relname,
            struct dir_attr *attr_list, const struct cp_options *x)
{
struct dir_attr *p;
char *dst_name;           /* A copy of CONST_DST_NAME we can change. */
ASSIGN_STRDUPA (dst_name, const_dst_name);

/* The suffix of DST_NAME that is a copy of the source file name,
possibly truncated to name a parent directory. */
char const *src_name = dst_name + (dst_src_name - const_dst_name);

/* Likewise, but with any leading '/'s skipped. */
char const *relname = dst_name + (dst_relname - const_dst_name);

for (p = attr_list; p; p = p->next)
{
    dst_name[p->slash_offset] = '\0';

/* Adjust the times (and if possible, ownership) for the copy.
   chown turns off set[ug]id bits for non-root,
   so do the chmod last. */

```

```

if (x->preserve_timestamps)
{
    struct timespec timespec[2];

    timespec[0] = get_stat_atime (&p->st);
    timespec[1] = get_stat_mtime (&p->st);

    if (utimensat (dst_dirfd, relname, timespec, 0))
    {
        error (0, errno, _("failed to preserve times for %s"),
               quoteaf (dst_name));
        return false;
    }
}

if (x->preserve_ownership)
{
    if (lchownat (dst_dirfd, relname, p->st.st_uid, p->st.st_gid)
        != 0)
    {
        if (! chown_failure_ok (x))
        {
            error (0, errno, _("failed to preserve ownership for %s"),
                   quoteaf (dst_name));
            return false;
        }
        /* Failing to preserve ownership is OK. Still, try to preserve
           the group, but ignore the possible error.*/
        ignore_value (lchownat (dst_dirfd, relname, -1, p->st.st_gid));
    }
}

if (x->preserve_mode)
{
    if (copy_acl (src_name, -1, dst_name, -1, p->st.st_mode) != 0)
        return false;
}
else if (p->restore_mode)
{
    if (lchmodat (dst_dirfd, relname, p->st.st_mode) != 0)
    {
        error (0, errno, _("failed to preserve permissions for %s"),
               quoteaf (dst_name));
        return false;
    }
}

dst_name[p->slash_offset] = '/';
}

return true;
}

/* Ensure that the parent directory of CONST_DIR exists, for
   the --parents option.

```

SRC\_OFFSET is the index in CONST\_DIR (which is a destination directory) of the beginning of the source directory name.  
Create any leading directories that don't already exist.  
DST\_DIRFD is a file descriptor for the target directory.

If VERBOSE\_FMT\_STRING is nonzero, use it as a printf format string for printing a message after successfully making a directory.

The format should take two string arguments: the names of the source and destination directories.

Creates a linked list of attributes of intermediate directories, \*ATTR\_LIST, for re\_protect to use after calling copy.

Sets \*NEW\_DST if this function creates parent of CONST\_DIR.

Return true if parent of CONST\_DIR exists as a directory with the proper permissions when done. \*/

```

/* FIXME: Synch this function with the one in ../lib/mkdir-p.c. */

static bool
make_dir_parents_private (char const *const_dir, size_t src_offset,
                          int dst_dirfd,
                          char const *verbose_fmt_string,
                          struct dir_attr **attr_list, bool *new_dst,
                          const struct cp_options *x)
{
    struct stat stats;
    char *dir;           /* A copy of CONST_DIR we can change. */
    char *src;           /* Source name in DIR. */
    char *dst_dir; /* Leading directory of DIR. */
    idx_t dirlen = dir_len (const_dir);

    *attr_list = nullptr;

    /* Succeed immediately if the parent of CONST_DIR must already exist,
       as the target directory has already been checked. */
    if (dirlen <= src_offset)
        return true;

    ASSIGN_STRDUPA (dir, const_dir);

    src = dir + src_offset;

    dst_dir = alloca (dirlen + 1);
    memcpy (dst_dir, dir, dirlen);
    dst_dir[dirlen] = '\0';
    char const *dst_reldir = dst_dir + src_offset;
    while (*dst_reldir == '/')
        dst_reldir++;

    /* XXX: If all dirs are present at the destination,
       no permissions or security contexts will be updated. */
    if (fstatat (dst_dirfd, dst_reldir, &stats, 0) != 0)
    {
        /* A parent of CONST_DIR does not exist.
           Make all missing intermediate directories. */
        char *slash;

        slash = src;
        while (*slash == '/')
            slash++;
        dst_reldir = slash;

        while ((slash = strchr (slash, '/')))
        {
            struct dir_attr *new;

```

```

bool missing_dir;

*slash = '\0';
missing_dir = fstatat (dst_dirfd, dst_reldir, &stats, 0) != 0;

if (missing_dir || x->preserve_ownership || x->preserve_mode
    || x->preserve_timestamps)
{
    /* Add this directory to the list of directories whose
       modes might need fixing later. */
    struct stat src_st;
    int src_errno = (stat (src, &src_st) != 0
                    ? errno
                    : S_ISDIR (src_st.st_mode)
                    ? 0
                    : ENOTDIR);
    if (src_errno)
    {
        error (0, src_errno, _("failed to get attributes of %s"),
               quoteaf (src));
        return false;
    }

    new = xmalloc (sizeof *new);
    new->st = src_st;
    new->slash_offset = slash - dir;
    new->restore_mode = false;
    new->next = *attr_list;
    *attr_list = new;
}

/* If required set the default context for created dirs. */
if (!set_process_security_ctx (src, dir,
                               missing_dir ? new->st.st_mode : 0,
                               missing_dir, x))
    return false;

if (missing_dir)
{
    mode_t src_mode;
    mode_t omitted_permissions;
    mode_t mkdir_mode;

    /* This component does not exist. We must set
       *new_dst and new->st.st_mode inside this loop because,
       for example, in the command 'cp --parents ..//a//b/c e_dir',
       make_dir_parents_private creates only e_dir//a if
       ./b already exists. */
    *new_dst = true;
    src_mode = new->st.st_mode;

    /* If the ownership or special mode bits might change,
       omit some permissions at first, so unauthorized users
       cannot nap in before the file is ready. */
    omitted_permissions = (src_mode
                           & (x->preserve_ownership
                               ? S_IRWXG | S_IRWXO
                               : x->preserve_mode
                               ? S_IWGRP | S_IWOTH
                               : 0));
}

```

```

/* POSIX says mkdir's behavior is implementation-defined when
   (src_mode & ~S_IRWXUGO) != 0. However, common practice is
   to ask mkdir to copy all the CHMOD_MODE_BITS, letting mkdir
   decide what to do with S_ISUID | S_ISGID | S_ISVTX. */
mkdir_mode = x->explicit_no_preserve_mode ? S_IRWXUGO : src_mode;
mkdir_mode &= CHMOD_MODE_BITS & ~omitted_permissions;
if (mkdirat (dst_dirfd, dst_reldir, mkdir_mode) != 0)
{
    error (0, errno, _("cannot make directory %s"),
           quoteaf (dir));
    return false;
}
else
{
    if (verbose_fmt_string != nullptr)
        printf (verbose_fmt_string, src, dir);
}

/* We need search and write permissions to the new directory
   for writing the directory's contents. Check if these
   permissions are there. */

if (fstatat (dst_dirfd, dst_reldir, &stats, AT_SYMLINK_NOFOLLOW))
{
    error (0, errno, _("failed to get attributes of %s"),
           quoteaf (dir));
    return false;
}

if (!x->preserve_mode)
{
    if (omitted_permissions & ~stats.st_mode)
        omitted_permissions &= ~ cached_umask ();
    if (omitted_permissions & ~stats.st_mode
        || (stats.st_mode & S_IRWXU) != S_IRWXU)
    {
        new->st.st_mode = stats.st_mode | omitted_permissions;
        new->restore_mode = true;
    }
}

mode_t accessible = stats.st_mode | S_IRWXU;
if (stats.st_mode != accessible)
{
    /* Make the new directory searchable and writable.
       The original permissions will be restored later. */

    if (!chmodat (dst_dirfd, dst_reldir, accessible) != 0)
    {
        error (0, errno, _("setting permissions for %s"),
               quoteaf (dir));
        return false;
    }
}
else if (!S_ISDIR (stats.st_mode))
{
    error (0, 0, ("%s exists but is not a directory"),

```

```

        quoteaf (dir));
    return false;
}
else
    *new_dst = false;

/* For existing dirs, set the security context as per that already
   set for the process global context. */
if (! *new_dst
    && (x->set_security_context || x->preserve_security_context))
{
    if (! set_file_security_ctx (dir, false, x)
        && x->require_preserve_context)
        return false;
}

*slash++ = '/';

/* Avoid unnecessary calls to 'stat' when given
   file names containing multiple adjacent slashes. */
while (*slash == '/')
    slash++;
}

/* We get here if the parent of DIR already exists. */

else if (!S_ISDIR (stats.st_mode))
{
    error (0, 0, _("'%s exists but is not a directory"), quoteaf (dst_dir));
    return false;
}
else
{
    *new_dst = false;
}
return true;
}

/* Scan the arguments, and copy each by calling copy.
   Return true if successful. */

static bool
do_copy (int n_files, char **file, char const *target_directory,
         bool no_target_directory, struct cp_options *x)
{
struct stat sb;
bool new_dst = false;
bool ok = true;

if (n_files <= !target_directory)
{
    if (n_files <= 0)
        error (0, 0, _("missing file operand"));
    else
        error (0, 0, _("missing destination file operand after %s"),
               quoteaf (file[0]));
    usage (EXIT_FAILURE);
}

```

```

sb.st_mode = 0;
int target_dirfd = AT_FDCWD;
if (!no_target_directory)
{
    if (target_directory)
        error (EXIT_FAILURE, 0,
               _("cannot combine --target-directory (-t) "
                 "and --no-target-directory (-T)"));
    if (2 < n_files)
    {
        error (0, 0, _("extra operand %s"), quoteaf (file[2]));
        usage (EXIT_FAILURE);
    }
}
else if (target_directory)
{
    target_dirfd = target_directory_operand (target_directory, &sb);
    if (!target_dirfd_valid (target_dirfd))
        error (EXIT_FAILURE, errno, _("target directory %s"),
               quoteaf (target_directory));
}
else
{
    char const *lastfile = file[n_files - 1];
    int fd = target_directory_operand (lastfile, &sb);
    if (target_dirfd_valid (fd))
    {
        target_dirfd = fd;
        target_directory = lastfile;
        n_files--;
    }
    else
    {
        int err = errno;
        if (err == ENOENT)
            new_dst = true;

        /* The last operand LASTFILE cannot be opened as a directory.
         * If there are more than two operands, report an error.
         */

        Also, report an error if LASTFILE is known to be a directory
        even though it could not be opened, which can happen if
        opening failed with EACCES on a platform lacking O_PATH.
        In this case use stat to test whether LASTFILE is a
        directory, in case opening a non-directory with (O_SEARCH
        | O_DIRECTORY) failed with EACCES not ENOTDIR. */
        if (2 < n_files
            || (O_PATHSEARCH == O_SEARCH && err == EACCES
                && (sb.st_mode || stat (lastfile, &sb) == 0)
                && S_ISDIR (sb.st_mode)))
            error (EXIT_FAILURE, err, _("target %s"), quoteaf (lastfile));
    }
}

if (target_directory)
{
    /* cp file1...filen edir
     * Copy the files 'file1' through 'filen'
     * to the existing directory 'edir'. */
}

```

```

/* Initialize these hash tables only if we'll need them.
   The problems they're used to detect can arise only if
   there are two or more files to copy. */
if (2 <= n_files)
{
    {
        dest_info_init (x);
        src_info_init (x);
    }

for (int i = 0; i < n_files; i++)
{
    {
        char *dst_name;
        bool parent_exists = true; /* True if dir_name (dst_name) exists. */
        struct dir_attr *attr_list;
        char *arg_in_concat;
        char *arg = file[i];

        /* Trailing slashes are meaningful (i.e., maybe worth preserving)
           only in the source file names. */
        if (remove_trailing_slashes)
            strip_trailing_slashes (arg);

        if (parents_option)
        {
            char *arg_no_trailing_slash;

            /* Use 'arg' without trailing slashes in constructing destination
               file names. Otherwise, we can end up trying to create a
               directory using a name with trailing slash, which fails on
               NetBSD 1.[34] systems. */
            ASSIGN_STRDUPA (arg_no_trailing_slash, arg);
            strip_trailing_slashes (arg_no_trailing_slash);

            /* Append all of 'arg' (minus any trailing slash) to 'dest'. */
            dst_name = file_name_concat (target_directory,
                                         arg_no_trailing_slash,
                                         &arg_in_concat);

            /* For --parents, we have to make sure that the directory
               dir_name (dst_name) exists. We may have to create a few
               leading directories. */
            parent_exists =
                (make_dir_parents_private
                 (dst_name, arg_in_concat - dst_name, target_dirfd,
                  (x->verbose ? "%s -> %s\n" : nullptr),
                  &attr_list, &new_dst, x));
        }
        else
        {
            char *arg_base;
            /* Append the last component of 'arg' to 'target_directory'. */
            ASSIGN_STRDUPA (arg_base, last_component (arg));
            strip_trailing_slashes (arg_base);
            /* For 'cp -R source/.. dest', don't copy into 'dest/..'. */
            arg_base += STREQ (arg_base, "..");
            dst_name = file_name_concat (target_directory, arg_base,
                                         &arg_in_concat);
        }
    }

    if (!parent_exists)

```

```

{
/* make_dir_parents_private failed, so don't even
   attempt the copy. */
ok = false;
}
else
{
char const *dst_relname = arg_in_concat;
while (*dst_relname == '/')
    dst_relname++;

bool copy_into_self;
ok &= copy (arg, dst_name, target_dirfd, dst_relname,
            new_dst, x, &copy_into_self, nullptr);

if (parents_option)
    ok &= re_protect (dst_name, arg_in_concat, target_dirfd,
                      dst_relname, attr_list, x);
}

if (parents_option)
{
while (attr_list)
{
    struct dir_attr *p = attr_list;
    attr_list = attr_list->next;
    free (p);
}
}

free (dst_name);
}

else /* !target_directory */
{
char const *source = file[0];
char const *dest = file[1];
bool unused;

if (parents_option)
{
    error (0, 0,
           _("with --parents, the destination must be a directory"));
    usage (EXIT_FAILURE);
}

/* When the force and backup options have been specified and
   the source and destination are the same name for an existing
   regular file, convert the user's command, e.g.,
   'cp --force --backup foo foo' to 'cp --force foo fooSUFFIX'
   where SUFFIX is determined by any version control options used. */

if (x->unlink_dest_after_failed_open
    && x->backup_type != no_backups
    && STREQ (source, dest)
    && !new_dst
    && (sb.st_mode != 0 || stat (dest, &sb) == 0) && S_ISREG (sb.st_mode))
{
static struct cp_options x_tmp;

```

```

dest = find_backup_file_name (AT_FDCWD, dest, x->backup_type);
/* Set x->backup_type to 'no_backups' so that the normal backup
   mechanism is not used when performing the actual copy.
   backup_type must be set to 'no_backups' only *after* the above
   call to find_backup_file_name -- that function uses
   backup_type to determine the suffix it applies. */
x_tmp = *x;
x_tmp.backup_type = no_backups;
x = &x_tmp;
}

ok = copy (source, dest, AT_FDCWD, dest, -new_dst, x, &unused, nullptr);
}

return ok;
}

static void
cp_option_init (struct cp_options *x)
{
cp_options_default (x);
x->copy_as_regular = true;
x->dereference = DEREF_UNDEFINED;
x->unlink_dest_before_opening = false;
x->unlink_dest_after_failed_open = false;
x->hard_link = false;
x->interactive = I_UNSPECIFIED;
x->move_mode = false;
x->install_mode = false;
x->one_file_system = false;
x->reflink_mode = REFLINK_AUTO;

x->preserve_ownership = false;
x->preserve_links = false;
x->preserve_mode = false;
x->preserve_timestamps = false;
x->explicit_no_preserve_mode = false;
x->preserve_security_context = false; /* -a or --preserve=context. */
x->require_preserve_context = false; /* --preserve=context. */
x->set_security_context = nullptr; /* -Z, set sys default context. */
x->preserve_xattr = false;
x->reduce_diagnostics = false;
x->require_preserve_xattr = false;

x->data_copy_required = true;
x->require_preserve = false;
x->recursive = false;
x->sparse_mode = SPARSE_AUTO;
x->symbolic_link = false;
x->set_mode = false;
x->mode = 0;

/* Not used. */
x->stdin_tty = false;

x->update = false;
x->verbose = false;
x->keep_directory_symlink = false;

/* By default, refuse to open a dangling destination symlink, because

```

```

in general one cannot do that safely, give the current semantics of
open's O_EXCL flag, (which POSIX doesn't even allow cp to use, btw).
But POSIX requires it. */
x->open_dangling_dest_symlink = getenv ("POSIXLY_CORRECT") != nullptr;

x->dest_info = nullptr;
x->src_info = nullptr;
}

/* Given a string, ARG, containing a comma-separated list of arguments
to the --preserve option, set the appropriate fields of X to ON_OFF. */
static void
decode_preserve_arg (char const *arg, struct cp_options *x, bool on_off)
{
enum File_attribute
{
    PRESERVE_MODE,
    PRESERVE_TIMESTAMPS,
    PRESERVE_OWNERSHIP,
    PRESERVE_LINK,
    PRESERVE_CONTEXT,
    PRESERVE_XATTR,
    PRESERVE_ALL
};
static enum File_attribute const preserve_vals[] =
{
    PRESERVE_MODE, PRESERVE_TIMESTAMPS,
    PRESERVE_OWNERSHIP, PRESERVE_LINK, PRESERVE_CONTEXT,
PRESERVE_XATTR,
    PRESERVE_ALL
};
/* Valid arguments to the '--preserve' option. */
static char const *const preserve_args[] =
{
    "mode", "timestamps",
    "ownership", "links", "context", "xattr", "all", nullptr
};
ARGMATCH_VERIFY (preserve_args, preserve_vals);

char *arg_writable = xstrdup (arg);
char *s = arg_writable;
do
{
    /* find next comma */
    char *comma = strchr (s, ',');
    enum File_attribute val;

    /* If we found a comma, put a NUL in its place and advance. */
    if (comma)
        *comma++ = 0;

    /* process S. */
    val = XARGMATCH (on_off ? "--preserve" : "--no-preserve",
                     s, preserve_args, preserve_vals);
    switch (val)
    {
        case PRESERVE_MODE:
            x->preserve_mode = on_off;
            x->explicit_no_preserve_mode = !on_off;
            break;
    }
}

```

```

    case PRESERVE_TIMESTAMPS:
        x->preserve_timestamps = on_off;
        break;

    case PRESERVE_OWNERSHIP:
        x->preserve_ownership = on_off;
        break;

    case PRESERVE_LINK:
        x->preserve_links = on_off;
        break;

    case PRESERVE_CONTEXT:
        x->require_preserve_context = on_off;
        x->preserve_security_context = on_off;
        break;

    case PRESERVE_XATTR:
        x->preserve_xattr = on_off;
        x->require_preserve_xattr = on_off;
        break;

    case PRESERVE_ALL:
        x->preserve_mode = on_off;
        x->preserve_timestamps = on_off;
        x->preserve_ownership = on_off;
        x->preserve_links = on_off;
        x->explicit_no_preserve_mode = !on_off;
        if (selinux_enabled)
            x->preserve_security_context = on_off;
        x->preserve_xattr = on_off;
        break;

    default:
        affirm (false);
    }
    s = comma;
}
while (s);

free (arg_writable);
}

int
main (int argc, char **argv)
{
int c;
bool ok;
bool make_backups = false;
char const *backup_suffix = nullptr;
char *version_control_string = nullptr;
struct cp_options x;
bool copy_contents = false;
char *target_directory = nullptr;
bool no_target_directory = false;
char const *scontext = nullptr;
bool no_clobber = false;

initialize_main (&argc, &argv);

```

```

set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

atexit (close_stdin);

selinux_enabled = (0 < is_selinux_enabled ());
cp_option_init (&x);

while ((c = getopt_long (argc, argv, "abdfHilLnprst:uvxPRS:TZ",
                       long_opts, nullptr))
      != -1)
{
switch (c)
{
case SPARSE_OPTION:
    x.sparse_mode = XARGMATCH ("--sparse", optarg,
                               sparse_type_string, sparse_type);
break;

case REFLINK_OPTION:
    if (optarg == nullptr)
        x.reflink_mode = REFLINK_ALWAYS;
    else
        x.reflink_mode = XARGMATCH ("--reflink", optarg,
                                   reflink_type_string, reflink_type);
break;

case 'a':
/* Like -dR --preserve=all with reduced failure diagnostics. */
    x.dereference = DEREF_NEVER;
    x.preserve_links = true;
    x.preserve_ownership = true;
    x.preserve_mode = true;
    x.preserve_timestamps = true;
    x.require_preserve = true;
    if (selinux_enabled)
        x.preserve_security_context = true;
    x.preserve_xattr = true;
    x.reduce_diagnostics = true;
    x.recursive = true;
break;

case 'b':
    make_backups = true;
    if (optarg)
        version_control_string = optarg;
break;

case ATTRIBUTES_ONLY_OPTION:
    x.data_copy_required = false;
break;

case DEBUG_OPTION:
    x.debug = x.verbose = true;
break;

case COPY_CONTENTS_OPTION:
    copy_contents = true;
}

```

```
break;

case 'd':
x.preserve_links = true;
x.dereference = DEREF_NEVER;
break;

case 'f':
x.unlink_dest_after_failed_open = true;
break;

case 'H':
x.dereference = DEREF_COMMAND_LINE_ARGUMENTS;
break;

case 'i':
x.interactive = I_ASK_USER;
break;

case 'l':
x.hard_link = true;
break;

case 'L':
x.dereference = DEREF_ALWAYS;
break;

case 'n':
x.interactive = I_ALWAYS_SKIP;
no_clobber = true;
x.update = false;
break;

case 'P':
x.dereference = DEREF_NEVER;
break;

case NO_PRESERVE_ATTRIBUTES_OPTION:
decode_preserve_arg (optarg, &x, false);
break;

case PRESERVE_ATTRIBUTES_OPTION:
if (optarg == nullptr)
{
    /* Fall through to the case for 'p' below. */
}
else
{
    decode_preserve_arg (optarg, &x, true);
    x.require_preserve = true;
    break;
}
FALLTHROUGH;

case 'p':
x.preserve_ownership = true;
x.preserve_mode = true;
x.preserve_timestamps = true;
x.require_preserve = true;
break;
```

```

case PARENTS_OPTION:
parents_option = true;
break;

case 'r':
case 'R':
x.recursive = true;
break;

case UNLINK_DEST_BEFORE_OPENING:
x.unlink_dest_before_opening = true;
break;

case STRIP_TRAILING_SLASHES_OPTION:
remove_trailing_slashes = true;
break;

case 's':
x.symbolic_link = true;
break;

case 't':
if (target_directory)
    error (EXIT_FAILURE, 0,
           _("multiple target directories specified"));
target_directory = optarg;
break;

case 'T':
no_target_directory = true;
break;

case 'u':
if (! no_clobber) /* -n > -u */
{
    enum Update_type update_opt = UPDATE_OLDER;
    if (optarg)
        update_opt = XARGMATCH ("--update", optarg,
                               update_type_string, update_type);
    if (update_opt == UPDATE_ALL)
    {
        /* Default cp operation. */
        x.update = false;
        x.interactive = I_UNSPECIFIED;
    }
    else if (update_opt == UPDATE_NONE)
    {
        x.update = false;
        x.interactive = I_ALWAYS_SKIP;
    }
    else if (update_opt == UPDATE_NONE_FAIL)
    {
        x.update = false;
        x.interactive = I_ALWAYS_NO;
    }
    else if (update_opt == UPDATE_OLDER)
    {
        x.update = true;
        x.interactive = I_UNSPECIFIED;
    }
}

```

```

        }
    }

break;

case 'v':
x.verbose = true;
break;

case KEEP_DIRECTORY_SYMLINK_OPTION:
x.keep_directory_symlink = true;
break;

case 'x':
x.one_file_system = true;
break;

case 'Z':
/* politely decline if we're not on a selinux-enabled kernel. */
if (selinux_enabled)
{
    if (optarg)
        scontext = optarg;
    else
    {
        x.set_security_context = selabel_open (SELABEL_CTX_FILE,
                                              nullptr, 0);
        if (! x.set_security_context)
            error (0, errno, _("warning: ignoring --context"));
    }
}
else if (optarg)
{
    error (0, 0,
           _("warning: ignoring --context; "
             "it requires an SELinux-enabled kernel"));
}
break;

case 'S':
make_backups = true;
backup_suffix = optarg;
break;

case_GETOPT_HELP_CHAR;

case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

default:
usage (EXIT_FAILURE);
}
}

/*
With --sparse=never, disable reflinking so we create a non sparse copy.
This will also have the effect of disabling copy offload as that may
propagate holes. For e.g. FreeBSD documents that copy_file_range()
will try to propagate holes. */
if (x.reflink_mode == REFLINK_AUTO && x.sparse_mode == SPARSE_NEVER)
    x.reflink_mode = REFLINK_NEVER;

if (x.hard_link && x.symbolic_link)

```

```

{
error (0, 0, _("cannot make both hard and symbolic links"));
usage (EXIT_FAILURE);
}

if (make_backups
    && (x.interactive == I_ALWAYS_SKIP
        || x.interactive == I_ALWAYS_NO))
{
error (0, 0,
      _("--backup is mutually exclusive with -n or --update=none-fail"));
usage (EXIT_FAILURE);
}

if (x.reflink_mode == REFLINK_ALWAYS && x.sparse_mode != SPARSE_AUTO)
{
error (0, 0, _("--reflink can be used only with --sparse=auto"));
usage (EXIT_FAILURE);
}

x.backup_type = (make_backups
                  ? xget_version (_("backup type"),
                                  version_control_string)
                  : no_backups);
set_simple_backup_suffix (backup_suffix);

if (x.dereference == DREF_UNDEFINED)
{
if (x.recursive && ! x.hard_link)
    /* This is compatible with FreeBSD. */
    x.dereference = DREF_NEVER;
else
    x.dereference = DREF_ALWAYS;
}

if (x.recursive)
    x.copy_as_regular = copy_contents;

/* Ensure -Z overrides -a. */
if ((x.set_security_context || scontext)
    && ! x.require_preserve_context)
    x.preserve_security_context = false;

if (x.preserve_security_context && (x.set_security_context || scontext))
    error (EXIT_FAILURE, 0,
          _("cannot set target context and preserve it"));

if (x.require_preserve_context && ! selinux_enabled)
    error (EXIT_FAILURE, 0,
          _("cannot preserve security context "
            "without an SELinux-enabled kernel"));

/* FIXME: This handles new files. But what about existing files?
I.e., if updating a tree, new files would have the specified context,
but shouldn't existing files be updated for consistency like this?
if (scontext && !restorecon (nullptr, dst_path, 0))
    error (...);
*/
if (scontext && setfscreatecon (scontext) < 0)
    error (EXIT_FAILURE, errno,

```

```

        _("failed to set default file creation context to %s"),
        quote (scontext);

#if !USE_XATTR
if (x.require_preserve_xattr)
    error (EXIT_FAILURE, 0, _("cannot preserve extended attributes, cp is "
                           "built without xattr support"));
#endif

/* Allocate space for remembering copied and created files. */

hash_init ();

ok = do_copy (argc - optind, argv + optind,
              target_directory, no_target_directory, &x);

main_exit (ok ? EXIT_SUCCESS : EXIT_FAILURE);
}

""",
"error_category": " z/OS-Specific Runtime Error ",
"error": """
EDC5129I: Cannot fork child process due to system limitations.
EDC5120I: Attempt to execute /bin/cp failed, possibly due to incorrect path handling in z/
OS.

""",
"correct_code": """
/* cp.c -- file copying (main routines)
Copyright (C) 1989-2024 Free Software Foundation, Inc.


```

This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License  
along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Written by Torbjörn Granlund, David MacKenzie, and Jim Meyering. \*/

```

#include <config.h>
#include <stdio.h>
#include <sys/types.h>
#include <getopt.h>
#include <selinux/label.h>

#include "system.h"
#include "argmatch.h"
#include "assure.h"
#include "backupfile.h"
#include "copy.h"
#include "cp-hash.h"
#include "filenamecat.h"
#include "ignore-value.h"

```

```

#include "quote.h"
#include "stat-time.h"
#include "targetdir.h"
#include "utimens.h"
#include "acl.h"

/* The official name of this program (e.g., no 'g' prefix). */
#define PROGRAM_NAME "cp"

#define AUTHORS \
proper_name_lite ("Torbjorn Granlund", "Torbj\303\266rn Granlund"), \
proper_name ("David MacKenzie"), \
proper_name ("Jim Meyering")

/* Used by do_copy, make_dir_parents_private, and re_protect
to keep a list of leading directories whose protections
need to be fixed after copying. */
struct dir_attr
{
    struct stat st;
    bool restore_mode;
    size_t slash_offset;
    struct dir_attr *next;
};

/* For long options that have no equivalent short option, use a
non-character as a pseudo short option, starting with CHAR_MAX + 1. */
enum
{
    ATTRIBUTES_ONLY_OPTION = CHAR_MAX + 1,
    COPY_CONTENTS_OPTION,
    DEBUG_OPTION,
    NO_PRESERVE_ATTRIBUTES_OPTION,
    PARENTS_OPTION,
    PRESERVE_ATTRIBUTES_OPTION,
    REFLINK_OPTION,
    SPARSE_OPTION,
    STRIP_TRAILING_SLASHES_OPTION,
    UNLINK_DEST_BEFORE_OPENING,
    KEEP_DIRECTORY_SYMLINK_OPTION
};

/* True if the kernel is SELinux enabled. */
static bool selinux_enabled;

/* If true, the command "cp x/e_file e_dir" uses "e_dir/x/e_file"
as its destination instead of the usual "e_dir/e_file." */
static bool parents_option = false;

/* Remove any trailing slashes from each SOURCE argument. */
static bool remove_trailing_slashes;

static char const *const sparse_type_string[] =
{
    "never", "auto", "always", nullptr
};
static enum Sparse_type const sparse_type[] =
{
    SPARSE_NEVER, SPARSE_AUTO, SPARSE_ALWAYS
};

```

```

ARGMATCH_VERIFY (sparse_type_string, sparse_type);

static char const *const reflink_type_string[] =
{
"auto", "always", "never", nullptr
};
static enum Reflink_type const reflink_type[] =
{
REFLINK_AUTO, REFLINK_ALWAYS, REFLINK_NEVER
};
ARGMATCH_VERIFY (reflink_type_string, reflink_type);

static char const *const update_type_string[] =
{
"all", "none", "older", nullptr
};
static enum Update_type const update_type[] =
{
UPDATE_ALL, UPDATE_NONE, UPDATE_OLDER,
};
ARGMATCH_VERIFY (update_type_string, update_type);

static struct option const long_opts[] =
{
{"archive", no_argument, nullptr, 'a'},
 {"attributes-only", no_argument, nullptr, ATTRIBUTES_ONLY_OPTION},
 {"backup", optional_argument, nullptr, 'b'},
 {"copy-contents", no_argument, nullptr, COPY_CONTENTS_OPTION},
 {"debug", no_argument, nullptr, DEBUG_OPTION},
 {"dereference", no_argument, nullptr, 'L'},
 {"force", no_argument, nullptr, 'f'},
 {"interactive", no_argument, nullptr, 'i'},
 {"link", no_argument, nullptr, 'l'},
 {"no-clobber", no_argument, nullptr, 'n'}, /* Deprecated. */
 {"no-dereference", no_argument, nullptr, 'P'},
 {"no-preserve", required_argument, nullptr, NO_PRESERVE_ATTRIBUTES_OPTION},
 {"no-target-directory", no_argument, nullptr, 'T'},
 {"one-file-system", no_argument, nullptr, 'x'},
 {"parents", no_argument, nullptr, PARENTS_OPTION},
 {"path", no_argument, nullptr, PARENTS_OPTION}, /* Deprecated. */
 {"preserve", optional_argument, nullptr, PRESERVE_ATTRIBUTES_OPTION},
 {"recursive", no_argument, nullptr, 'R'},
 {"remove-destination", no_argument, nullptr, UNLINK_DEST_BEFORE_OPENING},
 {"sparse", required_argument, nullptr, SPARSE_OPTION},
 {"reflink", optional_argument, nullptr, REFLINK_OPTION},
 {"strip-trailing-slashes", no_argument, nullptr,
STRIP_TRAILING_SLASHES_OPTION},
 {"suffix", required_argument, nullptr, 'S'},
 {"symbolic-link", no_argument, nullptr, 's'},
 {"target-directory", required_argument, nullptr, 't'},
 {"update", optional_argument, nullptr, 'u'},
 {"verbose", no_argument, nullptr, 'v'},
 {"keep-directory-symlink", no_argument, nullptr,
KEEP_DIRECTORY_SYMLINK_OPTION},
 {GETOPT_SELINUX_CONTEXT_OPTION_DECL},
 {GETOPT_HELP_OPTION_DECL},
 {GETOPT_VERSION_OPTION_DECL},
 {nullptr, 0, nullptr, 0}
};

```

```

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    {
        printf (_("\
Usage: %s [OPTION]... [-T] SOURCE DEST\n\
or: %s [OPTION]... SOURCE... DIRECTORY\n\
or: %s [OPTION]... -t DIRECTORY SOURCE...\\n\
"),
                program_name, program_name, program_name);
        fputs (_("\
Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.\\n\
"), stdout);
    }
    emit_mandatory_arg_note ();

    fputs (_("\
-a, --archive      same as -dR --preserve=all\\n\
--attributes-only   don't copy the file data, just the attributes\\n\
--backup[=CONTROL]   make a backup of each existing destination file\\
\\n\
-b                  like --backup but does not accept an argument\\n\
--copy-contents     copy contents of special files when recursive\\n\
-d                  same as --no-dereference --preserve=links\\n\
"),
            stdout);
    fputs (_("\
--debug             explain how a file is copied. Implies -v\\n\
"),
            stdout);
    fputs (_("\
-f, --force          if an existing destination file cannot be\\n\
                     opened, remove it and try again (this option\\n\
                     is ignored when the -n option is also used)\\n\
                     prompt before overwrite (overrides a previous -n\\
\\n\
                     option)\\n\
-H                  follow command-line symbolic links in SOURCE\\n\
"),
            stdout);
    fputs (_("\
-l, --link            hard link files instead of copying\\n\
-L, --dereference     always follow symbolic links in SOURCE\\n\
"),
            stdout);
    fputs (_("\
-n, --no-clobber      (deprecated) silently skip existing files.\\n\
                     See also --update\\n\
"),
            stdout);
    fputs (_("\
-P, --no-dereference   never follow symbolic links in SOURCE\\n\
"),
            stdout);
    fputs (_("\
-p                  same as --preserve=mode,ownership,timestamps\\n\
--preserve[=ATTR_LIST]  preserve the specified attributes\\n\
"),
            stdout);
    fputs (_("\
--no-preserve=ATTR_LIST  don't preserve the specified attributes\\n\
--parents              use full source file name under DIRECTORY\\n\
"),
            stdout);
    fputs (_("\
"),
            stdout);
}

```

```

-R, -r, --recursive      copy directories recursively\n\
    --reflink[=WHEN]      control clone/CoW copies. See below\n\
    --remove-destination  remove each existing destination file before\n\
                           attempting to open it (contrast with --force)\n\
\n"), stdout);
    fputs (_("\
    --sparse=WHEN        control creation of sparse files. See below\n\
    --strip-trailing-slashes remove any trailing slashes from each SOURCE\n\
                           argument\n\
"), stdout);
    fputs (_("\
    -s, --symbolic-link  make symbolic links instead of copying\n\
    -S, --suffix=SUFFIX   override the usual backup suffix\n\
    -t, --target-directory=DIRECTORY copy all SOURCE arguments into DIRECTORY\n\
    -T, --no-target-directory  treat DEST as a normal file\n\
"), stdout);
    fputs (_("\
    --update[=UPDATE]     control which existing files are updated;\n\
                           UPDATE={all,none,none-fail,older(default)}.\n\
    -u                     equivalent to --update[=older]. See below\n\
"), stdout);
    fputs (_("\
    -v, --verbose         explain what is being done\n\
"), stdout);
    fputs (_("\
    --keep-directory-symlink follow existing symlinks to directories\n\
"), stdout);
    fputs (_("\
    -x, --one-file-system stay on this file system\n\
"), stdout);
    fputs (_("\
    -Z                   set SELinux security context of destination\n\
                           file to default type\n\
    --context[=CTX]       like -Z, or if CTX is specified then set the\n\
                           SELinux or SMACK security context to CTX\n\
"), stdout);
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
    fputs (_("\
\n\
ATTR_LIST is a comma-separated list of attributes. Attributes are 'mode' for\n\
permissions (including any ACL and xattr permissions), 'ownership' for user\n\
and group, 'timestamps' for file timestamps, 'links' for hard links, 'context'\n\
for security context, 'xattr' for extended attributes, and 'all' for all\n\
attributes.\n\
"), stdout);
    fputs (_("\
\n\
By default, sparse SOURCE files are detected by a crude heuristic and the\n\
corresponding DEST file is made sparse as well. That is the behavior\n\
selected by --sparse=auto. Specify --sparse=always to create a sparse DEST\n\
file whenever the SOURCE file contains a long enough sequence of zero bytes.\n\
Use --sparse=never to inhibit creation of sparse files.\n\
"), stdout);
    emit_update_parameters_note ();
    fputs (_("\
\n\
When --reflink[=always] is specified, perform a lightweight copy, where the\n\
data blocks are copied only when modified. If this is not possible the copy\n\
fails, or if --reflink=auto is specified, fall back to a standard copy.\n\

```

```

Use --reflink=never to ensure a standard copy is performed.\n\
"), stdout);
    emit_backup_suffix_note ();
    fputs (_("\
\n\
As a special case, cp makes a backup of SOURCE when the force and backup\n\
options are given and SOURCE and DEST are the same name for an existing,\n\
regular file.\n\
"), stdout);
    emit_ancillary_info (PROGRAM_NAME);
}
exit (status);
}

/* Ensure that parents of CONST_DST_NAME have correct protections, for
the --parents option. This is done after all copying has been
completed, to allow permissions that don't include user write/execute.

```

DST\_SRC\_NAME is the suffix of CONST\_DST\_NAME that is the source file name,  
DST\_DIRFD+DST\_RELNAME is equivalent to CONST\_DST\_NAME, and  
DST\_RELNAME equals DST\_SRC\_NAME after skipping any leading '/'s.

ATTR\_LIST is a null-terminated linked list of structures that  
indicates the end of the filename of each intermediate directory  
in CONST\_DST\_NAME that may need to have its attributes changed.  
The command 'cp --parents --preserve a/b/c d/e\_dir' changes the  
attributes of the directories d/e\_dir/a and d/e\_dir/a/b to match  
the corresponding source directories regardless of whether they  
existed before the 'cp' command was given.

Return true if the parent of CONST\_DST\_NAME and any intermediate  
directories specified by ATTR\_LIST have the proper permissions  
when done. \*/

```

static bool
re_protect (char const *const_dst_name, char const *dst_src_name,
            int dst_dirfd, char const *dst_relname,
            struct dir_attr *attr_list, const struct cp_options *x)
{
struct dir_attr *p;
char *dst_name;           /* A copy of CONST_DST_NAME we can change. */
ASSIGN_STRDUPA (dst_name, const_dst_name);

/* The suffix of DST_NAME that is a copy of the source file name,
possibly truncated to name a parent directory. */
char const *src_name = dst_name + (dst_src_name - const_dst_name);

/* Likewise, but with any leading '/'s skipped. */
char const *relname = dst_name + (dst_relname - const_dst_name);

for (p = attr_list; p; p = p->next)
{
    dst_name[p->slash_offset] = '\0';

/* Adjust the times (and if possible, ownership) for the copy.
chown turns off set[ug]id bits for non-root,
so do the chmod last. */

if (x->preserve_timestamps)

```

```

{
struct timespec timespec[2];

timespec[0] = get_stat_atime (&p->st);
timespec[1] = get_stat_mtime (&p->st);

if (utimensat (dst_dirfd, relname, timespec, 0))
{
    error (0, errno, _("failed to preserve times for %s"),
           quoteaf (dst_name));
    return false;
}
}

if (x->preserve_ownership)
{
    if (!chownat (dst_dirfd, relname, p->st.st_uid, p->st.st_gid)
        != 0)
    {
        if (! chown_failure_ok (x))
        {
            error (0, errno, _("failed to preserve ownership for %s"),
                   quoteaf (dst_name));
            return false;
        }
        /* Failing to preserve ownership is OK. Still, try to preserve
           the group, but ignore the possible error. */
        ignore_value (lchownat (dst_dirfd, relname, -1, p->st.st_gid));
    }
}

if (x->preserve_mode)
{
    if (copy_acl (src_name, -1, dst_name, -1, p->st.st_mode) != 0)
        return false;
}
else if (p->restore_mode)
{
    if (!chmodat (dst_dirfd, relname, p->st.st_mode) != 0)
    {
        error (0, errno, _("failed to preserve permissions for %s"),
               quoteaf (dst_name));
        return false;
    }
}

dst_name[p->slash_offset] = '/';
}

return true;
}

/* Ensure that the parent directory of CONST_DIR exists, for
the --parents option.

```

SRC\_OFFSET is the index in CONST\_DIR (which is a destination directory) of the beginning of the source directory name.  
Create any leading directories that don't already exist.  
DST\_DIRFD is a file descriptor for the target directory.  
If VERBOSE\_FMT\_STRING is nonzero, use it as a printf format string for printing a message after successfully making a directory.

The format should take two string arguments: the names of the source and destination directories.  
Creates a linked list of attributes of intermediate directories,  
\*ATTR\_LIST, for re\_protect to use after calling copy.  
Sets \*NEW\_DST if this function creates parent of CONST\_DIR.  
Return true if parent of CONST\_DIR exists as a directory with the proper permissions when done. \*/

```
/* FIXME: Synch this function with the one in ../lib/mkdir-p.c. */

static bool
make_dir_parents_private (char const *const_dir, size_t src_offset,
                         int dst_dirfd,
                         char const *verbose_fmt_string,
                         struct dir_attr **attr_list, bool *new_dst,
                         const struct cp_options *x)
{
    struct stat stats;
    char *dir;           /* A copy of CONST_DIR we can change. */
    char *src;           /* Source name in DIR. */
    char *dst_dir;       /* Leading directory of DIR. */
    idx_t dirlen = dir_len (const_dir);

    *attr_list = nullptr;

    /* Succeed immediately if the parent of CONST_DIR must already exist,
       as the target directory has already been checked. */
    if (dirlen <= src_offset)
        return true;

    ASSIGN_STRDUPA (dir, const_dir);

    src = dir + src_offset;

    dst_dir = alloca (dirlen + 1);
    memcpy (dst_dir, dir, dirlen);
    dst_dir[dirlen] = '\0';
    char const *dst_reldir = dst_dir + src_offset;
    while (*dst_reldir == '/')
        dst_reldir++;

    /* XXX: If all dirs are present at the destination,
       no permissions or security contexts will be updated. */
    if (fstatat (dst_dirfd, dst_reldir, &stats, 0) != 0)
    {
        /* A parent of CONST_DIR does not exist.
           Make all missing intermediate directories. */
        char *slash;

        slash = src;
        while (*slash == '/')
            slash++;
        dst_reldir = slash;

        while ((slash = strchr (slash, '/')))
        {
            struct dir_attr *new;
            bool missing_dir;
```

```

*slash = '\0';
missing_dir = fstatat (dst_dirfd, dst_reldir, &stats, 0) != 0;

if (missing_dir || x->preserve_ownership || x->preserve_mode
    || x->preserve_timestamps)
{
    /* Add this directory to the list of directories whose
       modes might need fixing later. */
    struct stat src_st;
    int src_errno = (stat (src, &src_st) != 0
                     ? errno
                     : S_ISDIR (src_st.st_mode)
                     ? 0
                     : ENOTDIR);
    if (src_errno)
    {
        error (0, src_errno, _("failed to get attributes of %s"),
               quoteaf (src));
        return false;
    }

    new = xmalloc (sizeof *new);
    new->st = src_st;
    new->slash_offset = slash - dir;
    new->restore_mode = false;
    new->next = *attr_list;
    *attr_list = new;
}

/* If required set the default context for created dirs. */
if (!set_process_security_ctx (src, dir,
                               missing_dir ? new->st.st_mode : 0,
                               missing_dir, x))
    return false;

if (missing_dir)
{
    mode_t src_mode;
    mode_t omitted_permissions;
    mode_t mkdir_mode;

    /* This component does not exist. We must set
       *new_dst and new->st.st_mode inside this loop because,
       for example, in the command 'cp --parents ..//a//b/c e_dir',
       make_dir_parents_private creates only e_dir//a if
       ./b already exists. */
    *new_dst = true;
    src_mode = new->st.st_mode;

    /* If the ownership or special mode bits might change,
       omit some permissions at first, so unauthorized users
       cannot nip in before the file is ready. */
    omitted_permissions = (src_mode
                           & (x->preserve_ownership
                               ? S_IRWXG | S_IRWXO
                               : x->preserve_mode
                               ? S_IWGRP | S_IWOTH
                               : 0));
}

/* POSIX says mkdir's behavior is implementation-defined when

```

```

(src_mode & ~S_IRWXUGO) != 0. However, common practice is
to ask mkdir to copy all the CHMOD_MODE_BITS, letting mkdir
decide what to do with S_ISUID | S_ISGID | S_ISVTX. */
mkdir_mode = x->explicit_no_preserve_mode ? S_IRWXUGO : src_mode;
mkdir_mode &= CHMOD_MODE_BITS & ~omitted_permissions;
if (mkdirat (dst_dirfd, dst_reldir, mkdir_mode) != 0)
{
    error (0, errno, _("cannot make directory %s"),
           quoteaf (dir));
    return false;
}
else
{
    if (verbose_fmt_string != nullptr)
        printf (verbose_fmt_string, src, dir);
}
/* We need search and write permissions to the new directory
   for writing the directory's contents. Check if these
   permissions are there. */

if (fstatat (dst_dirfd, dst_reldir, &stats, AT_SYMLINK_NOFOLLOW))
{
    error (0, errno, _("failed to get attributes of %s"),
           quoteaf (dir));
    return false;
}

if (!x->preserve_mode)
{
    if (omitted_permissions & ~stats.st_mode)
        omitted_permissions &= ~cached_umask ();
    if (omitted_permissions & ~stats.st_mode
        || (stats.st_mode & S_IRWXU) != S_IRWXU)
    {
        new->st.st_mode = stats.st_mode | omitted_permissions;
        new->restore_mode = true;
    }
}

mode_t accessible = stats.st_mode | S_IRWXU;
if (stats.st_mode != accessible)
{
    /* Make the new directory searchable and writable.
       The original permissions will be restored later. */

    if (lchmodat (dst_dirfd, dst_reldir, accessible) != 0)
    {
        error (0, errno, _("setting permissions for %s"),
               quoteaf (dir));
        return false;
    }
}
else if (!S_ISDIR (stats.st_mode))
{
    error (0, 0, ("%s exists but is not a directory"),
           quoteaf (dir));
    return false;
}

```

```

        }
    else
        *new_dst = false;

    /* For existing dirs, set the security context as per that already
       set for the process global context. */
    if (! *new_dst
        && (x->set_security_context || x->preserve_security_context))
    {
        if (! set_file_security_ctx (dir, false, x)
            && x->require_preserve_context)
            return false;
    }

    *slash++ = '/';

    /* Avoid unnecessary calls to 'stat' when given
       file names containing multiple adjacent slashes. */
    while (*slash == '/')
        slash++;
}

/* We get here if the parent of DIR already exists. */

else if (!S_ISDIR (stats.st_mode))
{
    error (0, 0, _("'%s exists but is not a directory"), quoteaf (dst_dir));
    return false;
}
else
{
    *new_dst = false;
}
return true;
}

#ifndef __MVS__
#define MVS_COPY "/bin/cp"
#define MVS_COPY_LEN (sizeof(MVS_COPY)-1)
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
static bool
do_mvs_copy (int argc, char **argv)
{
    pid_t pid = fork();
    if (pid == 0) {
        argv[0] = MVS_COPY;
        execvp(MVS_COPY, argv);
    } else if (pid > 0) {
        int wstatus;
        pid_t waitchild = waitpid(pid, &wstatus, WUNTRACED | WCONTINUED);
        if (waitchild == -1) {
            perror("waitpid");
            return false;
        }
        if (WIFEXITED(wstatus)) {
            return WEXITSTATUS(wstatus) == 0 ? true : false;
        }
    }
}

```

```

        } else if (WIFSIGNALED(wstatus)) {
            fprintf(stderr, "killed by signal %d\n", WTERMSIG(wstatus));
        }
    }
#endif

/* Scan the arguments, and copy each by calling copy.
Return true if successful. */

static bool
do_copy (int n_files, char **file, char const *target_directory,
         bool no_target_directory, struct cp_options *x)
{
    struct stat sb;
    bool new_dst = false;
    bool ok = true;

    if (n_files <= !target_directory)
    {
        if (n_files <= 0)
            error (0, 0, _("missing file operand"));
        else
            error (0, 0, _("missing destination file operand after %s"),
                  quoteaf (file[0]));
        usage (EXIT_FAILURE);
    }

    sb.st_mode = 0;
    int target_dirfd = AT_FDCWD;
    if (no_target_directory)
    {
        if (target_directory)
            error (EXIT_FAILURE, 0,
                  _("cannot combine --target-directory (-t) "
                    "and --no-target-directory (-T)"));
        if (2 < n_files)
        {
            error (0, 0, _("extra operand %s"), quoteaf (file[2]));
            usage (EXIT_FAILURE);
        }
    }
    else if (target_directory)
    {
        target_dirfd = target_directory_operand (target_directory, &sb);
        if (!target_dirfd_valid (target_dirfd))
            error (EXIT_FAILURE, errno, _("target directory %s"),
                  quoteaf (target_directory));
    }
    else
    {
        char const *lastfile = file[n_files - 1];
        int fd = target_directory_operand (lastfile, &sb);
        if (target_dirfd_valid (fd))
        {
            target_dirfd = fd;
            target_directory = lastfile;
            n_files--;
        }
    }
}

```

```

{
int err = errno;
if (err == ENOENT)
    new_dst = true;

/* The last operand LASTFILE cannot be opened as a directory.
If there are more than two operands, report an error.

Also, report an error if LASTFILE is known to be a directory
even though it could not be opened, which can happen if
opening failed with EACCES on a platform lacking O_PATH.
In this case use stat to test whether LASTFILE is a
directory, in case opening a non-directory with (O_SEARCH
| O_DIRECTORY) failed with EACCES not ENOTDIR. */
if (2 < n_files
    || (O_PATHSEARCH == O_SEARCH && err == EACCES
        && (sb.st_mode || stat (lastfile, &sb) == 0)
        && S_ISDIR (sb.st_mode)))
    error (EXIT_FAILURE, err, _("target %s"), quoteaf (lastfile));
}

if (target_directory)
{
/* cp file1...filen edir
Copy the files 'file1' through 'filen'
to the existing directory 'edir'. */

/* Initialize these hash tables only if we'll need them.
The problems they're used to detect can arise only if
there are two or more files to copy. */
if (2 <= n_files)
{
    dest_info_init (x);
    src_info_init (x);
}

for (int i = 0; i < n_files; i++)
{
    char *dst_name;
    bool parent_exists = true; /* True if dir_name (dst_name) exists. */
    struct dir_attr *attr_list;
    char *arg_in_concat;
    char *arg = file[i];

    /* Trailing slashes are meaningful (i.e., maybe worth preserving)
       only in the source file names. */
    if (remove_trailing_slashes)
        strip_trailing_slashes (arg);

    if (parents_option)
    {
        char *arg_no_trailing_slash;

        /* Use 'arg' without trailing slashes in constructing destination
           file names. Otherwise, we can end up trying to create a
           directory using a name with trailing slash, which fails on
           NetBSD 1.[34] systems. */
        ASSIGN_STRDUPA (arg_no_trailing_slash, arg);
        strip_trailing_slashes (arg_no_trailing_slash);
    }
}
}

```

```

/* Append all of 'arg' (minus any trailing slash) to 'dest'. */
dst_name = file_name_concat (target_directory,
                            arg_no_trailing_slash,
                            &arg_in_concat);

/* For --parents, we have to make sure that the directory
   dir_name (dst_name) exists. We may have to create a few
   leading directories.*/
parent_exists =
    (make_dir_parents_private
     (dst_name, arg_in_concat - dst_name, target_dirfd,
      (x->verbose ? "%s -> %s\n" : nullptr),
      &attr_list, &new_dst, x));
}
else
{
    char *arg_base;
    /* Append the last component of 'arg' to 'target_directory'. */
    ASSIGN_STRDUPA (arg_base, last_component (arg));
    strip_trailing_slashes (arg_base);
    /* For 'cp -R source/.. dest', don't copy into 'dest/..'. */
    arg_base += STREQ (arg_base, "..");
    dst_name = file_name_concat (target_directory, arg_base,
                                 &arg_in_concat);
}

if (!parent_exists)
{
    /* make_dir_parents_private failed, so don't even
       attempt the copy. */
    ok = false;
}
else
{
    char const *dst_relname = arg_in_concat;
    while (*dst_relname == '/')
        dst_relname++;

    bool copy_into_self;
    ok &= copy (arg, dst_name, target_dirfd, dst_relname,
                new_dst, x, &copy_into_self, nullptr);

    if (parents_option)
        ok &= re_protect (dst_name, arg_in_concat, target_dirfd,
                          dst_relname, attr_list, x);
}

if (parents_option)
{
    while (attr_list)
    {
        struct dir_attr *p = attr_list;
        attr_list = attr_list->next;
        free (p);
    }
}

free (dst_name);
}

```

```

        }

else /* !target_directory */
{
    char const *source = file[0];
    char const *dest = file[1];
    bool unused;

    if (parents_option)
    {
        error (0, 0,
               _("with --parents, the destination must be a directory"));
        usage (EXIT_FAILURE);
    }

/* When the force and backup options have been specified and
   the source and destination are the same name for an existing
   regular file, convert the user's command, e.g.,
   'cp --force --backup foo foo' to 'cp --force foo fooSUFFIX'
   where SUFFIX is determined by any version control options used. */

if (x->unlink_dest_after_failed_open
    && x->backup_type != no_backups
    && STREQ (source, dest)
    && !new_dst
    && (sb.st_mode != 0 || stat (dest, &sb) == 0) && S_ISREG (sb.st_mode))
{
    static struct cp_options x_tmp;

    dest = find_backup_file_name (AT_FDCWD, dest, x->backup_type);
    /* Set x->backup_type to 'no_backups' so that the normal backup
       mechanism is not used when performing the actual copy.
       backup_type must be set to 'no_backups' only *after* the above
       call to find_backup_file_name -- that function uses
       backup_type to determine the suffix it applies. */
    x_tmp = *x;
    x_tmp.backup_type = no_backups;
    x = &x_tmp;
}

ok = copy (source, dest, AT_FDCWD, dest, -new_dst, x, &unused, nullptr);
}

return ok;
}

static void
cp_option_init (struct cp_options *x)
{
    cp_options_default (x);
    x->copy_as_regular = true;
    x->dereference = DEREF_UNDEFINED;
    x->unlink_dest_before_opening = false;
    x->unlink_dest_after_failed_open = false;
    x->hard_link = false;
    x->interactive = I_UNSPECIFIED;
    x->move_mode = false;
    x->install_mode = false;
    x->one_file_system = false;
    x->reflink_mode = REFLINK_AUTO;
}

```

```

x->preserve_ownership = false;
x->preserve_links = false;
x->preserve_mode = false;
x->preserve_timestamps = false;
x->explicit_no_preserve_mode = false;
x->preserve_security_context = false; /* -a or --preserve=context. */
x->require_preserve_context = false; /* --preserve=context. */
x->set_security_context = nullptr; /* -Z, set sys default context. */
x->preserve_xattr = false;
x->reduce_diagnostics = false;
x->require_preserve_xattr = false;

x->data_copy_required = true;
x->require_preserve = false;
x->recursive = false;
x->sparse_mode = SPARSE_AUTO;
x->symbolic_link = false;
x->set_mode = false;
x->mode = 0;

/* Not used. */
x->stdin_tty = false;

x->update = false;
x->verbose = false;
x->keep_directory_symlink = false;

/* By default, refuse to open a dangling destination symlink, because
   in general one cannot do that safely, give the current semantics of
   open's O_EXCL flag, (which POSIX doesn't even allow cp to use, btw).
   But POSIX requires it. */
x->open_dangling_dest_symlink = getenv ("POSIXLY_CORRECT") != nullptr;

x->dest_info = nullptr;
x->src_info = nullptr;
}

/* Given a string, ARG, containing a comma-separated list of arguments
   to the --preserve option, set the appropriate fields of X to ON/OFF. */
static void
decode_preserve_arg (char const *arg, struct cp_options *x, bool on_off)
{
enum File_attribute
{
  PRESERVE_MODE,
  PRESERVE_TIMESTAMPS,
  PRESERVE_OWNERSHIP,
  PRESERVE_LINK,
  PRESERVE_CONTEXT,
  PRESERVE_XATTR,
  PRESERVE_ALL
};
static enum File_attribute const preserve_vals[] =
{
  PRESERVE_MODE, PRESERVE_TIMESTAMPS,
  PRESERVE_OWNERSHIP, PRESERVE_LINK, PRESERVE_CONTEXT,
PRESERVE_XATTR,
  PRESERVE_ALL
};
/* Valid arguments to the '--preserve' option. */

```

```

static char const *const preserve_args[] =
{
    "mode", "timestamps",
    "ownership", "links", "context", "xattr", "all", nullptr
};
ARGMATCH_VERIFY (preserve_args, preserve_vals);

char *arg_writable = xstrdup (arg);
char *s = arg_writable;
do
{
    /* find next comma */
    char *comma = strchr (s, ',');
    enum File_attribute val;

    /* If we found a comma, put a NUL in its place and advance. */
    if (comma)
        *comma++ = 0;

    /* process S. */
    val = XARGMATCH (on_off ? "--preserve" : "--no-preserve",
                     s, preserve_args, preserve_vals);
    switch (val)
    {
        case PRESERVE_MODE:
            x->preserve_mode = on_off;
            x->explicit_no_preserve_mode = !on_off;
            break;

        case PRESERVE_TIMESTAMPS:
            x->preserve_timestamps = on_off;
            break;

        case PRESERVE_OWNERSHIP:
            x->preserve_ownership = on_off;
            break;

        case PRESERVE_LINK:
            x->preserve_links = on_off;
            break;

        case PRESERVE_CONTEXT:
            x->require_preserve_context = on_off;
            x->preserve_security_context = on_off;
            break;

        case PRESERVE_XATTR:
            x->preserve_xattr = on_off;
            x->require_preserve_xattr = on_off;
            break;

        case PRESERVE_ALL:
            x->preserve_mode = on_off;
            x->preserve_timestamps = on_off;
            x->preserve_ownership = on_off;
            x->preserve_links = on_off;
            x->explicit_no_preserve_mode = !on_off;
            if (selinux_enabled)
                x->preserve_security_context = on_off;
            x->preserve_xattr = on_off;
    }
}

```

```

break;

default:
    affirm (false);
}
s = comma;
}
while (s);

free (arg_writable);
}

int
main (int argc, char **argv)
{
int c;
bool ok;
bool make_backups = false;
char const *backup_suffix = nullptr;
char *version_control_string = nullptr;
struct cp_options x;
bool copy_contents = false;
char *target_directory = nullptr;
bool no_target_directory = false;
char const *scontext = nullptr;
bool no_clobber = false;

#ifndef __MVS__
{
    int i;

    for (i=0; i<argc; ++i) {
        if (argv[i][0] == '/' && argv[i][1] == '/') {
            ok = do_mvs_copy(argc, argv);
            main_exit (ok ? EXIT_SUCCESS : EXIT_FAILURE);
        }
    }
}
#endif

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

atexit (close_stdin);

selinux_enabled = (0 < is_selinux_enabled ());
cp_option_init (&x);

while ((c = getopt_long (argc, argv, "abdfHilLnprst:uvxPRS:TZ",
                        long_opts, nullptr))
      != -1)
{
switch (c)
{
case SPARSE_OPTION:
    x.sparse_mode = XARGMATCH ("--sparse", optarg,
                               sparse_type_string, sparse_type);
}
}
}

```

```
break;

case REFLINK_OPTION:
if (optarg == nullptr)
    x.reflink_mode = REFLINK_ALWAYS;
else
    x.reflink_mode = XARGMATCH ("--reflink", optarg,
                                reflink_type_string, reflink_type);
break;

case 'a':
/* Like -dR --preserve=all with reduced failure diagnostics. */
x.dereference = DEREF_NEVER;
x.preserve_links = true;
x.preserve_ownership = true;
x.preserve_mode = true;
x.preserve_timestamps = true;
x.require_preserve = true;
if (selinux_enabled)
    x.preserve_security_context = true;
x.preserve_xattr = true;
x.reduce_diagnostics = true;
x.recursive = true;
break;

case 'b':
make_backups = true;
if (optarg)
    version_control_string = optarg;
break;

case ATTRIBUTES_ONLY_OPTION:
x.data_copy_required = false;
break;

case DEBUG_OPTION:
x.debug = x.verbose = true;
break;

case COPY_CONTENTS_OPTION:
copy_contents = true;
break;

case 'd':
x.preserve_links = true;
x.dereference = DEREF_NEVER;
break;

case 'f':
x.unlink_dest_after_failed_open = true;
break;

case 'H':
x.dereference = DEREF_COMMAND_LINE_ARGUMENTS;
break;

case 'i':
x.interactive = I_ASK_USER;
break;
```

```
case 'I':
x.hard_link = true;
break;

case 'L':
x.dereference = DEREF_ALWAYS;
break;

case 'n':
x.interactive = I_ALWAYS_SKIP;
no_clobber = true;
x.update = false;
break;

case 'P':
x.dereference = DEREF_NEVER;
break;

case NO_PRESERVE_ATTRIBUTES_OPTION:
decode_preserve_arg (optarg, &x, false);
break;

case PRESERVE_ATTRIBUTES_OPTION:
if (optarg == nullptr)
{
    /* Fall through to the case for 'p' below. */
}
else
{
    decode_preserve_arg (optarg, &x, true);
    x.require_preserve = true;
    break;
}
FALLTHROUGH;

case 'p':
x.preserve_ownership = true;
x.preserve_mode = true;
x.preserve_timestamps = true;
x.require_preserve = true;
break;

case PARENTS_OPTION:
parents_option = true;
break;

case 'r':
case 'R':
x.recursive = true;
break;

case UNLINK_DEST_BEFORE_OPENING:
x.unlink_dest_before_opening = true;
break;

case STRIP_TRAILING_SLASHES_OPTION:
remove_trailing_slashes = true;
break;

case 's':
```

```

x.symbolic_link = true;
break;

case 't':
if (target_directory)
    error (EXIT_FAILURE, 0,
           _("multiple target directories specified"));
target_directory = optarg;
break;

case 'T':
no_target_directory = true;
break;

case 'u':
if (! no_clobber) /* -n > -u */
{
    enum Update_type update_opt = UPDATE_OLDER;
    if (optarg)
        update_opt = XARGMATCH ("--update", optarg,
                               update_type_string, update_type);
    if (update_opt == UPDATE_ALL)
    {
        /* Default cp operation. */
        x.update = false;
        x.interactive = I_UNSPECIFIED;
    }
    else if (update_opt == UPDATE_NONE)
    {
        x.update = false;
        x.interactive = I_ALWAYS_SKIP;
    }
    else if (update_opt == UPDATE_NONE_FAIL)
    {
        x.update = false;
        x.interactive = I_ALWAYS_NO;
    }
    else if (update_opt == UPDATE_OLDER)
    {
        x.update = true;
        x.interactive = I_UNSPECIFIED;
    }
}
break;

case 'v':
x.verbose = true;
break;

case KEEP_DIRECTORY_SYMLINK_OPTION:
x.keep_directory_symlink = true;
break;

case 'x':
x.one_file_system = true;
break;

case 'Z':
/* politely decline if we're not on a selinux-enabled kernel. */
if (selinux_enabled)

```

```

    {
        if (optarg)
            scontext = optarg;
        else
            {
                x.set_security_context = selabel_open (SELABEL_CTX_FILE,
                                                    nullptr, 0);
                if (!x.set_security_context)
                    error (0, errno, _("warning: ignoring --context"));
            }
    }
    else if (optarg)
    {
        error (0, 0,
               _("warning: ignoring --context; "
                 "it requires an SELinux-enabled kernel"));
    }
    break;

    case 'S':
        make_backups = true;
        backup_suffix = optarg;
        break;

    case_GETOPT_HELP_CHAR;

    case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

    default:
        usage (EXIT_FAILURE);
    }
}

/* With --sparse=never, disable reflinking so we create a non sparse copy.
   This will also have the effect of disabling copy offload as that may
   propagate holes. For e.g. FreeBSD documents that copy_file_range()
   will try to propagate holes. */
if (x.reflink_mode == REFLINK_AUTO && x.sparse_mode == SPARSE_NEVER)
    x.reflink_mode = REFLINK_NEVER;

if (x.hard_link && x.symbolic_link)
{
    error (0, 0, _("cannot make both hard and symbolic links"));
    usage (EXIT_FAILURE);
}

if (make_backups
    && (x.interactive == I_ALWAYS_SKIP
         || x.interactive == I_ALWAYS_NO))
{
    error (0, 0,
           _("--backup is mutually exclusive with -n or --update=none-fail"));
    usage (EXIT_FAILURE);
}

if (x.reflink_mode == REFLINK_ALWAYS && x.sparse_mode != SPARSE_AUTO)
{
    error (0, 0, _("--reflink can be used only with --sparse=auto"));
    usage (EXIT_FAILURE);
}

```

```

x.backup_type = (make_backups
    ? xget_version (_("backup type"),
                    version_control_string)
    : no_backups);
set_simple_backup_suffix (backup_suffix);

if (x.dereference == DEREF_UNDEFINED)
{
    if (x.recursive && ! x.hard_link)
        /* This is compatible with FreeBSD. */
        x.dereference = DEREF_NEVER;
    else
        x.dereference = DEREF_ALWAYS;
}

if (x.recursive)
    x.copy_as_regular = copy_contents;

/* Ensure -Z overrides -a. */
if ((x.set_security_context || scontext)
    && ! x.require_preserve_context)
    x.preserve_security_context = false;

if (x.preserve_security_context && (x.set_security_context || scontext))
    error (EXIT_FAILURE, 0,
           _("cannot set target context and preserve it"));

if (x.require_preserve_context && ! selinux_enabled)
    error (EXIT_FAILURE, 0,
           _("cannot preserve security context "
             "without an SELinux-enabled kernel"));

/* FIXME: This handles new files. But what about existing files?
I.e., if updating a tree, new files would have the specified context,
but shouldn't existing files be updated for consistency like this?
if (scontext && !restorecon (nullptr, dst_path, 0))
    error (...);
*/
if (scontext && setfscreatecon (scontext) < 0)
    error (EXIT_FAILURE, errno,
           _("failed to set default file creation context to %s"),
           quote (scontext));

#ifndef !USE_XATTR
if (x.require_preserve_xattr)
    error (EXIT_FAILURE, 0, _("cannot preserve extended attributes, cp is "
                           "built without xattr support"));
#endif

/* Allocate space for remembering copied and created files. */

hash_init ();

ok = do_copy (argc - optind, argv + optind,
              target_directory, no_target_directory, &x);

main_exit (ok ? EXIT_SUCCESS : EXIT_FAILURE);
}
"""
,
```

```

"patch":
"""

diff --git a/src/cp.c b/src/cp.c
index 28b0217..6bdc450 100644
--- a/src/cp.c
+++ b/src/cp.c
@@ -607,6 +607,36 @@ make_dir_parents_private (char const *const_dir, size_t src_offset,
return true;
}

+#ifdef __MVS__
+#define MVS_COPY "/bin/cp"
+#define MVS_COPY_LEN (sizeof(MVS_COPY)-1)
+#include <unistd.h>
+#include <sys/types.h>
+#include <sys/wait.h>
+static bool
+do_mvs_copy (int argc, char **argv)
+{
+
+ pid_t pid = fork();
+ if (pid == 0) {
+   argv[0] = MVS_COPY;
+   execvp(MVS_COPY, argv);
+ } else if (pid > 0) {
+   int wstatus;
+   pid_t waitchild = waitpid(pid, &wstatus, WUNTRACED | WCONTINUED);
+   if (waitchild == -1) {
+     perror("waitpid");
+     return false;
+   }
+   if (WIFEXITED(wstatus)) {
+     return WEXITSTATUS(wstatus) == 0 ? true : false;
+   } else if (WIFSIGNALED(wstatus)) {
+     fprintf(stderr, "killed by signal %d\n", WTERMSIG(wstatus));
+   }
+ }
+}
+#endif
+
/* Scan the arguments, and copy each by calling copy.
Return true if successful. */

@@ -986,6 +1016,19 @@ main (int argc, char **argv)
char const *scontext = nullptr;
bool no_clobber = false;

+#ifdef __MVS__
+
+ {
+   int i;
+
+   for (i=0; i<argc; ++i) {
+     if (argv[i][0] == '/' && argv[i][1] == '/') {
+       ok = do_mvs_copy(argc, argv);
+       main_exit (ok ? EXIT_SUCCESS : EXIT_FAILURE);
+     }
+   }
+ }
+#endif
+

```

```

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
"""
},
{
"wrong_code":
"""
/* Compute checksums of files or strings.
Copyright (C) 1995-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */

/* Written by Ulrich Drepper <drepper@gnu.ai.mit.edu>. */

#include <config.h>

#include <getopt.h>
#include <sys/types.h>

#include "system.h"
#include "argmatch.h"
#include "c-ctype.h"
#include "quote.h"
#include "xdecoint.h"
#include "xstrtol.h"

#ifndef HASH_ALGO_CKSUM
#define HASH_ALGO_CKSUM 0
#endif

#if HASH_ALGO_SUM || HASH_ALGO_CKSUM
#include "sum.h"
#endif
#if HASH_ALGO_CKSUM
#include "cksum.h"
#include "base64.h"
#endif
#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
#include "blake2/b2sum.h"
#endif
#if HASH_ALGO_MD5 || HASH_ALGO_CKSUM
#include "md5.h"
#endif
#if HASH_ALGO_SHA1 || HASH_ALGO_CKSUM
#include "sha1.h"
#endif
#if HASH_ALGO_SHA256 || HASH_ALGO_SHA224 || HASH_ALGO_CKSUM

```

```

# include "sha256.h"
#endif
#if HASH_ALGO_SHA512 || HASH_ALGO_SHA384 || HASH_ALGO_CKSUM
# include "sha512.h"
#endif
#if HASH_ALGO_CKSUM
# include "sm3.h"
#endif
#include "fadvise.h"
#include "stdio--.h"
#include "xbinary-io.h"

/* The official name of this program (e.g., no 'g' prefix). */
#ifndef PROGRAM_NAME
#define PROGRAM_NAME "sum"
#define DIGEST_TYPE_STRING "BSD"
#define DIGEST_STREAM sumfns[sum_algorithm]
#define DIGEST_OUT sum_output_fns[sum_algorithm]
#define DIGEST_BITS 16
#define DIGEST_ALIGN 4
#elif HASH_ALGO_CKSUM
#define MAX_DIGEST_BITS 512
#define MAX_DIGEST_ALIGN 8
#define PROGRAM_NAME "cksum"
#define DIGEST_TYPE_STRING algorithm_tags[cksum_algorithm]
#define DIGEST_STREAM cksumfns[cksum_algorithm]
#define DIGEST_OUT cksum_output_fns[cksum_algorithm]
#define DIGEST_BITS MAX_DIGEST_BITS
#define DIGEST_ALIGN MAX_DIGEST_ALIGN
#elif HASH_ALGO_MD5
#define PROGRAM_NAME "md5sum"
#define DIGEST_TYPE_STRING "MD5"
#define DIGEST_STREAM md5_stream
#define DIGEST_BITS 128
#define DIGEST_REFERENCE "RFC 1321"
#define DIGEST_ALIGN 4
#elif HASH_ALGO_BLAKE2
#define PROGRAM_NAME "b2sum"
#define DIGEST_TYPE_STRING "BLAKE2b"
#define DIGEST_STREAM blake2b_stream
#define DIGEST_BITS 512
#define DIGEST_REFERENCE "RFC 7693"
#define DIGEST_ALIGN 8
#elif HASH_ALGO_SHA1
#define PROGRAM_NAME "sha1sum"
#define DIGEST_TYPE_STRING "SHA1"
#define DIGEST_STREAM sha1_stream
#define DIGEST_BITS 160
#define DIGEST_REFERENCE "FIPS-180-1"
#define DIGEST_ALIGN 4
#elif HASH_ALGO_SHA256
#define PROGRAM_NAME "sha256sum"
#define DIGEST_TYPE_STRING "SHA256"
#define DIGEST_STREAM sha256_stream
#define DIGEST_BITS 256
#define DIGEST_REFERENCE "FIPS-180-2"
#define DIGEST_ALIGN 4
#elif HASH_ALGO_SHA224
#define PROGRAM_NAME "sha224sum"
#define DIGEST_TYPE_STRING "SHA224"

```

```

# define DIGEST_STREAM sha224_stream
# define DIGEST_BITS 224
# define DIGEST_REFERENCE "RFC 3874"
# define DIGEST_ALIGN 4
#elif HASH_ALGO_SHA512
# define PROGRAM_NAME "sha512sum"
# define DIGEST_TYPE_STRING "SHA512"
# define DIGEST_STREAM sha512_stream
# define DIGEST_BITS 512
# define DIGEST_REFERENCE "FIPS-180-2"
# define DIGEST_ALIGN 8
#elif HASH_ALGO_SHA384
# define PROGRAM_NAME "sha384sum"
# define DIGEST_TYPE_STRING "SHA384"
# define DIGEST_STREAM sha384_stream
# define DIGEST_BITS 384
# define DIGEST_REFERENCE "FIPS-180-2"
# define DIGEST_ALIGN 8
#else
# error "Can't decide which hash algorithm to compile."
#endif
#if !HASH_ALGO_SUM && !HASH_ALGO_CKSUM
# define DIGEST_OUT output_file
#endif

#if HASH_ALGO_SUM
# define AUTHORS \
proper_name ("Kayvan Aghaiepour"), \
proper_name ("David MacKenzie")
#elif HASH_ALGO_CKSUM
# define AUTHORS \
proper_name_lite ("Padraig Brady", "P\303\241draig Brady"), \
proper_name ("Q. Frank Xia")
#elif HASH_ALGO_BLAKE2
# define AUTHORS \
proper_name_lite ("Padraig Brady", "P\303\241draig Brady"), \
proper_name ("Samuel Neves")
#else
# define AUTHORS \
proper_name ("Ulrich Drepper"), \
proper_name ("Scott Miller"), \
proper_name ("David Madore")
#endif
#if !HASH_ALGO_BLAKE2 && !HASH_ALGO_CKSUM
# define DIGEST_HEX_BYTES (DIGEST_BITS / 4)
#endif
#define DIGEST_BIN_BYTES (DIGEST_BITS / 8)

/* The minimum length of a valid digest line. This length does
not include any newline character at the end of a line. */
#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
# define MIN_DIGEST_LINE_LENGTH 3 /* With -I 8. */
#else
# define MIN_DIGEST_LINE_LENGTH \
(DIGEST_HEX_BYTES /* length of hexadecimal message digest */ \
+ 1 /* blank */ \
+ 1 /* minimum filename length */ )
#endif

#if !HASH_ALGO_SUM

```

```

static void
output_file (char const *file, int binary_file, void const *digest,
             bool raw, bool tagged, unsigned char delim, bool args,
             uintmax_t length);
#endif

/* True if any of the files read were the standard input. */
static bool have_read_stdin;

/* The minimum length of a valid checksum line for the selected algorithm. */
static size_t min_digest_line_length;

/* Set to the length of a digest hex string for the selected algorithm. */
static size_t digest_hex_bytes;

/* With --check, don't generate any output.
The exit code indicates success or failure. */
static bool status_only = false;

/* With --check, print a message to standard error warning about each
improperly formatted checksum line. */
static bool warn = false;

/* With --check, ignore missing files. */
static bool ignore_missing = false;

/* With --check, suppress the "OK" printed for each verified file. */
static bool quiet = false;

/* With --check, exit with a non-zero return code if any line is
improperly formatted. */
static bool strict = false;

/* Whether a BSD reversed format checksum is detected. */
static int bsd_reversed = -1;

/* line delimiter. */
static unsigned char digest_delim = '\n';

#if HASH_ALGO_CKSUM
/* If true, print base64-encoded digests, not hex. */
static bool base64_digest = false;
#endif

/* If true, print binary digests, not hex. */
static bool raw_digest = false;

#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
#define BLAKE2B_MAX_LEN BLAKE2B_OUTBYTES
static uintmax_t digest_length;
#endif /* HASH_ALGO_BLAKE2 */

typedef void (*digest_output_fn)(char const *, int, void const *, bool,
                                bool, unsigned char, bool, uintmax_t);
#if HASH_ALGO_SUM
enum Algorithm
{
    bsd,
    sysv,
};

```

```

static enum Algorithm sum_algorithm;
static sumfn sumfns[]=
{
    bsd_sum_stream,
    sysv_sum_stream,
};
static digest_output_fn sum_output_fns[]=
{
    output_bsd,
    output_sysv,
};
#endif

#if HASH_ALGO_CKSUM
static int
md5_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return md5_stream (stream, resstream);
}
static int
sha1_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return sha1_stream (stream, resstream);
}
static int
sha224_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return sha224_stream (stream, resstream);
}
static int
sha256_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return sha256_stream (stream, resstream);
}
static int
sha384_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return sha384_stream (stream, resstream);
}
static int
sha512_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return sha512_stream (stream, resstream);
}
static int
blake2b_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return blake2b_stream (stream, resstream, *length);
}
static int
sm3_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return sm3_stream (stream, resstream);
}

enum Algorithm
{
    bsd,
    sysv,
}

```

```

crc,
md5,
sha1,
sha224,
sha256,
sha384,
sha512,
blake2b,
sm3,
};

static char const *const algorithm_args[] =
{
"bsd", "sysv", "crc", "md5", "sha1", "sha224",
"sha256", "sha384", "sha512", "blake2b", "sm3", nullptr
};
static enum Algorithm const algorithm_types[] =
{
bsd, sysv, crc, md5, sha1, sha224,
sha256, sha384, sha512, blake2b, sm3,
};
ARGMATCH_VERIFY(algorithm_args, algorithm_types);

static char const *const algorithm_tags[] =
{
"BSD", "SYSV", "CRC", "MD5", "SHA1", "SHA224",
"SHA256", "SHA384", "SHA512", "BLAKE2b", "SM3", nullptr
};
static int const algorithm_bits[] =
{
16, 16, 32, 128, 160, 224,
256, 384, 512, 512, 256, 0
};

static_assert(ARRAY_CARDINALITY(algorithm_bits)
    == ARRAY_CARDINALITY(algorithm_args));

static bool algorithm_specified = false;
static enum Algorithm cksum_algorithm = crc;
static sumfn cksumfns[]=
{
bsd_sum_stream,
sysv_sum_stream,
crc_sum_stream,
md5_sum_stream,
sha1_sum_stream,
sha224_sum_stream,
sha256_sum_stream,
sha384_sum_stream,
sha512_sum_stream,
blake2b_sum_stream,
sm3_sum_stream,
};
static digest_output_fn cksum_output_fns[]=
{
output_bsd,
output_sysv,
output_crc,
output_file,
output_file,

```

```

output_file,
output_file,
output_file,
output_file,
output_file,
output_file,
output_file,
};

bool cksum_debug;
#endif

/* For long options that have no equivalent short option, use a
non-character as a pseudo short option, starting with CHAR_MAX + 1. */

enum
{
IGNORE_MISSING_OPTION = CHAR_MAX + 1,
STATUS_OPTION,
QUIET_OPTION,
STRICT_OPTION,
TAG_OPTION,
UNTAG_OPTION,
DEBUG_PROGRAM_OPTION,
RAW_OPTION,
BASE64_OPTION,
};

static struct option const long_options[] =
{
#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
{ "length", required_argument, nullptr, 'l'},
#endif

#if !HASH_ALGO_SUM
{ "check", no_argument, nullptr, 'c' },
{ "ignore-missing", no_argument, nullptr, IGNORE_MISSING_OPTION},
{ "quiet", no_argument, nullptr, QUIET_OPTION },
{ "status", no_argument, nullptr, STATUS_OPTION },
{ "warn", no_argument, nullptr, 'w' },
{ "strict", no_argument, nullptr, STRICT_OPTION },
{ "tag", no_argument, nullptr, TAG_OPTION },
{ "zero", no_argument, nullptr, 'z' },
#endif

#if HASH_ALGO_CKSUM
{ "algorithm", required_argument, nullptr, 'a'},
{ "base64", no_argument, nullptr, BASE64_OPTION },
{ "debug", no_argument, nullptr, DEBUG_PROGRAM_OPTION},
{ "raw", no_argument, nullptr, RAW_OPTION},
{ "untagged", no_argument, nullptr, UNTAG_OPTION },
#endif

{ "binary", no_argument, nullptr, 'b' },
{ "text", no_argument, nullptr, 't' },

#else
{"sysv", no_argument, nullptr, 's'},
#endif

{ GETOPT_HELP_OPTION_DECL },
{ GETOPT_VERSION_OPTION_DECL },
{ nullptr, 0, nullptr, 0 }
};

```

```

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    {
        printf (_("\
Usage: %s [OPTION]... [FILE]...\\n\
"), program_name);
#if HASH_ALGO_CKSUM
    fputs (_("\
Print or verify checksums.\\n\
By default use the 32 bit CRC algorithm.\\n\
"), stdout);
#else
    printf (_("\
Print or check %s (%d-bit) checksums.\\n\
"),
            DIGEST_TYPE_STRING,
            DIGEST_BITS);
#endif

emit_stdin_note ();
#endif HASH_ALGO_SUM
fputs (_("\
\\n\
-r      use BSD sum algorithm (the default), use 1K blocks\\n\
-s, --sysv   use System V sum algorithm, use 512 bytes blocks\\n\
"), stdout);
#endif
#endif HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
emit_mandatory_arg_note ();
#endif
#endif HASH_ALGO_CKSUM
fputs (_("\
\\n\
-a, --algorithm=TYPE select the digest type to use. See DIGEST below.\\
\\n\
"), stdout);
fputs (_("\
--base64      emit base64-encoded digests, not hexadecimal\\
\\n\
"), stdout);
#endif
#endif !HASH_ALGO_SUM
#ifndef HASH_ALGO_CKSUM
if (O_BINARY)
    fputs (_("\
-b, --binary      read in binary mode (default unless reading tty stdin)\\
\\n\
"), stdout);
else
    fputs (_("\
-b, --binary      read in binary mode\\n\
"), stdout);
#endif
fputs (_("\
-c, --check      read checksums from the FILEs and check them\\n\
"), stdout);

```

```

# if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
    fputs (_("\
-l, --length=BITS    digest length in bits; must not exceed the max for\n\
                     the blake2 algorithm and must be a multiple of 8\n\
"), stdout);
#endif
#if HASH_ALGO_CKSUM
    fputs (_("\
--raw      emit a raw binary digest, not hexadecimal\n\
\n\
"), stdout);
    fputs (_("\
--tag      create a BSD-style checksum (the default)\n\
"), stdout);
    fputs (_("\
--untagged   create a reversed style checksum, without digest type\n\
"), stdout);
#else
    fputs (_("\
--tag      create a BSD-style checksum\n\
"), stdout);
#endif
#if !HASH_ALGO_CKSUM
    if (O_BINARY)
        fputs (_("\
-t, --text    read in text mode (default if reading tty stdin)\n\
"), stdout);
    else
        fputs (_("\
-t, --text    read in text mode (default)\n\
"), stdout);
#endif
    fputs (_("\
-z, --zero   end each output line with NUL, not newline,\n\
                     and disable file name escaping\n\
"), stdout);
    fputs (_("\
\n\
"), stdout);
The following five options are useful only when verifying checksums:\n\
    --ignore-missing  don't fail or report status for missing files\n\
    --quiet          don't print OK for each successfully verified file\n\
    --status         don't output anything, status code shows success\n\
    --strict         exit non-zero for improperly formatted checksum lines\n\
-w, --warn        warn about improperly formatted checksum lines\n\
\n\
"), stdout);
#endif
#if HASH_ALGO_CKSUM
    fputs (_("\
    --debug       indicate which implementation used\n\
"), stdout);
#endif
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
#endif
#if HASH_ALGO_CKSUM
    fputs (_("\
\n\
"), stdout);
DIGEST determines the digest algorithm and default output format:\n\
sysv  (equivalent to sum -s)\n\
bsd   (equivalent to sum -r)\n\

```

```

crc    (equivalent to cksum)\n\
md5    (equivalent to md5sum)\n\
sha1   (equivalent to sha1sum)\n\
sha224  (equivalent to sha224sum)\n\
sha256  (equivalent to sha256sum)\n\
sha384  (equivalent to sha384sum)\n\
sha512  (equivalent to sha512sum)\n\
blake2b (equivalent to b2sum)\n\
sm3    (only available through cksum)\n\
\n"), stdout);
#endif
#if !HASH_ALGO_SUM && !HASH_ALGO_CKSUM
    printf (_("\
\n")
The sums are computed as described in %s.\n"), DIGEST_REFERENCE);
    fputs (_("\
When checking, the input should be a former output of this program.\n\
The default mode is to print a line with: checksum, a space,\n\
a character indicating input mode ('*' for binary, ' ' for text\n\
or where binary is insignificant), and name for each FILE.\n\
\n\
There is no difference between binary mode and text mode on GNU systems.\n\
\n"), stdout);
#endif
#if HASH_ALGO_CKSUM
    fputs (_("\
When checking, the input should be a former output of this program,\n\
or equivalent standalone program.\n\
\n"), stdout);
#endif
    emit_ancillary_info (PROGRAM_NAME);
}

exit (status);
}

/* Given a string S, return TRUE if it contains problematic characters
that need escaping. Note we escape '\' itself to provide some forward
compat to introduce escaping of other characters. */
ATTRIBUTE_PURE
static bool
problematic_chars (char const *s)
{
size_t length = strcspn (s, "\\\n\r");
return s[length] != '\0';
}

#define ISWHITE(c) ((c) == ' ' || (c) == '\t')

/* Given a file name, S of length S_LEN, that is not NUL-terminated,
modify it in place, performing the equivalent of this sed substitution:
's\\n\\n/g;s\\r\\r/g;s\\\\\\Vg' i.e., replacing each "\n" string
with a newline, each "\\r" string with a carriage return,
and each "\\\\" with a single backslash, NUL-terminate it and return S.
If S is not a valid escaped file name, i.e., if it ends with an odd number
of backslashes or if it contains a backslash followed by anything other
than "n" or another backslash, return nullptr. */

static char *

```

```

filename_unescape (char *s, size_t s_len)
{
    char *dst = s;

    for (size_t i = 0; i < s_len; i++)
    {
        switch (s[i])
        {
            case '\\':
                if (i == s_len - 1)
                {
                    /* File name ends with an unescaped backslash: invalid. */
                    return nullptr;
                }
                ++i;
                switch (s[i])
                {
                    case 'n':
                        *dst++ = '\n';
                        break;
                    case 'r':
                        *dst++ = '\r';
                        break;
                    case '\\':
                        *dst++ = '\\';
                        break;
                    default:
                        /* Only '\', 'n' or 'r' may follow a backslash. */
                        return nullptr;
                }
                break;

            case '\0':
                /* The file name may not contain a NUL. */
                return nullptr;

            default:
                *dst++ = s[i];
                break;
        }
    }

    if (dst < s + s_len)
        *dst = '\0';

    return s;
}

/* Return true if S is a LEN-byte NUL-terminated string of hex or base64
digits and has the expected length. Otherwise, return false. */
ATTRIBUTE_PURE
static bool
valid_digits (unsigned char const *s, size_t len)
{
#ifndef HASH_ALGO_CKSUM
if (len == BASE64_LENGTH (digest_length / 8))
{
    size_t i;
    for (i = 0; i < len - digest_length % 3; i++)
    {
        if (!isbase64 (*s))

```

```

        return false;
    ++s;
}
for ( ; i < len; i++)
{
    if (*s != '=')
        return false;
    ++s;
}
}
else
#endif
if (len == digest_hex_bytes)
{
    for (idx_t i = 0; i < digest_hex_bytes; i++)
    {
        if (!lc_isxdigit (*s))
            return false;
        ++s;
    }
}
else
    return false;

return *s == '\0';
}

/* Split the checksum string S (of length S_LEN) from a BSD 'md5' or
'sha1' command into two parts: a hexadecimal digest, and the file
name. S is modified. Set *D_LEN to the length of the digest string.
Return true if successful. */

static bool
bsd_split_3 (char *s, size_t s_len,
             unsigned char **digest, size_t *d_len,
             char **file_name, bool escaped_filename)
{
if (s_len == 0)
    return false;

/* Find end of filename. */
size_t i = s_len - 1;
while (i && s[i] != ')')
    i--;

if (s[i] != ')')
    return false;

*file_name = s;

if (escaped_filename && filename_unescape (s, i) == nullptr)
    return false;

s[i++] = '\0';

while (ISWHITE (s[i]))
    i++;

if (s[i] != '=')
    return false;

```

```

i++;

while (ISWHITE (s[i]))
    i++;

*digest = (unsigned char *) &s[i];

*d_d_len = s_len - i;
return valid_digits (*digest, *d_len);
}

#if HASH_ALGO_CKSUM
/* Return the corresponding Algorithm for the string S,
or -1 for no match. */

static ptrdiff_t
algorithm_from_tag (char *s)
{
/* Limit check size to this length for perf reasons. */
static size_t max_tag_len;
if (!max_tag_len)
{
    char const * const * tag = algorithm_tags;
    while (*tag)
    {
        size_t tag_len = strlen (*tag++);
        max_tag_len = MAX (tag_len, max_tag_len);
    }
}
size_t i = 0;

/* Find end of tag */
while (i <= max_tag_len && s[i] && ! ISWHITE (s[i])
    && s[i] != '-' && s[i] != '(')
    ++i;

if (i > max_tag_len)
    return -1;

/* Terminate tag, and lookup. */
char sep = s[i];
s[i] = '\0';
ptrdiff_t algo = argmatch_exact (s, algorithm_tags);
s[i] = sep;

return algo;
}
#endif

/* Split the string S (of length S_LEN) into three parts:
a hexadecimal digest, binary flag, and the file name.
S is modified. Set *D_LEN to the length of the digest string.
Return true if successful. */

static bool
split_3 (char *s, size_t s_len,
         unsigned char **digest, size_t *d_len, int *binary, char **file_name)
{

```

```

bool escaped_filename = false;
size_t algo_name_len;

size_t i = 0;
while (ISWHITE (s[i]))
    ++i;

if (s[i] == '\\')
{
    ++
    escaped_filename = true;
}

/* Check for BSD-style checksum line. */

#if HASH_ALGO_CKSUM
if (!algorithm_specified)
{
    ptrdiff_t algo_tag = algorithm_from_tag (s + i);
    if (algo_tag >= 0)
    {
        if (algo_tag <= crc)
            return false; /* We don't support checking these older formats. */
        cksum_algorithm = algo_tag;
    }
    else
        return false; /* We only support tagged format without -a. */
}
#endif

algo_name_len = strlen (DIGEST_TYPE_STRING);
if (STREQ_LEN (s + i, DIGEST_TYPE_STRING, algo_name_len))
{
    i += algo_name_len;
#endif HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
/* Terminate and match algorithm name. */
char const *algo_name = &s[i - algo_name_len];
bool length_specified = s[i] == '-';
bool openssl_format = s[i] == '('; /* and no length_specified */
s[i++] = '\0';
if (!STREQ (algo_name, DIGEST_TYPE_STRING))
    return false;
if (openssl_format)
    s[--i] = '(';

#if HASH_ALGO_BLAKE2
    digest_length = BLAKE2B_MAX_LEN * 8;
#else
    digest_length = algorithm_bits[cksum_algorithm];
#endif
if (length_specified)
{
    uintmax_t length;
    char *siend;
    if (!(xstrtoumax (s + i, &siend, 0, &length, nullptr) == LONGINT_OK
          && 0 < length && length <= digest_length
          && length % 8 == 0))
        return false;
}

i = siend - s;

```

```

        digest_length = length;
    }
    digest_hex_bytes = digest_length / 4;
#endif
    if (s[i] == ' ')
        ++i;
    if (s[i] == '(')
    {
        ++
        *binary = 0;
        return bsd_split_3 (s + i, s_len - i,
                           digest, d_len, file_name, escaped_filename);
    }
    return false;
}

/* Ignore this line if it is too short.
   Each line must have at least 'min_digest_line_length - 1' (or one more, if
   the first is a backslash) more characters to contain correct message digest
   information. */
if (s_len - i < min_digest_line_length + (s[i] == '\\'))
    return false;

*digest = (unsigned char *) &s[i];

#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
/* Auto determine length. */
#ifndef HASH_ALGO_CKSUM
if (cksum_algorithm == blake2b) {
#endif
    unsigned char const *hp = *digest;
    digest_hex_bytes = 0;
    while (c_isxdigit (*hp++))
        digest_hex_bytes++;
    if (digest_hex_bytes < 2 || digest_hex_bytes % 2
        || BLAKE2B_MAX_LEN * 2 < digest_hex_bytes)
        return false;
    digest_length = digest_hex_bytes * 4;
#ifndef HASH_ALGO_CKSUM
}
#endif
#endif
#endif

/* This field must be the hexadecimal or base64 representation
   of the message digest. */
while (s[i] && !ISWHITE (s[i]))
    i++;

/* The digest must be followed by at least one whitespace character. */
if (i == s_len)
    return false;

*d_dlen = &s[i] - (char *) *digest;
s[i++] = '\0';

if (! valid_digits (*digest, *d_dlen))
    return false;

/* If "bsd reversed" format detected. */
if ((s_len - i == 1) || (s[i] != ' ' && s[i] != '*'))

```

```

{
/* Don't allow mixing bsd and standard formats,
   to minimize security issues with attackers
   renaming files with leading spaces.
   This assumes that with bsd format checksums
   that the first file name does not have
   a leading ' ' or '*' */
if (bsd_reversed == 0)
    return false;
bsd_reversed = 1;
}
else if (bsd_reversed != 1)
{
    bsd_reversed = 0;
    *binary = (s[i++] == '*');
}

/* All characters between the type indicator and end of line are
   significant -- that includes leading and trailing white space. */
*file_name = &s[i];

if (escaped_filename)
    return filename_unescape (&s[i], s_len - i) != nullptr;

return true;
}

/* If ESCAPE is true, then translate each:
   NEWLINE byte to the string, "\\n",
   CARRIAGE RETURN byte to the string, "\\r",
   and each backslash to "\\\\". */
static void
print_filename (char const *file, bool escape)
{
if (! escape)
{
    fputs (file, stdout);
    return;
}

while (*file)
{
switch (*file)
{
{
case '\n':
fputs ("\\n", stdout);
break;

case '\r':
fputs ("\\r", stdout);
break;

case '\\':
fputs ("\\\\\\", stdout);
break;

default:
putchar (*file);
break;
}
}

```

```

        file++;
    }
}

/* An interface to the function, DIGEST_STREAM.
Operate on FILENAME (it may be "-").

*BINAR indicates whether the file is binary. BINAR < 0 means it
depends on whether binary mode makes any difference and the file is
a terminal; in that case, clear *BINAR if the file was treated as
text because it was a terminal.

Put the checksum in *BIN_RESULT, which must be properly aligned.
Put true in *MISSING if the file can't be opened due to ENOENT.
Return true if successful. */

static bool
digest_file (char const *filename, int *binary, unsigned char *bin_result,
             bool *missing, MAYBE_UNUSED uintmax_t *length)
{
FILE *fp;
int err;
bool is_stdin = STREQ (filename, "-");

*missing = false;

if (is_stdin)
{
    have_read_stdin = true;
    fp = stdin;
    if (O_BINARY && *binary)
    {
        if (*binary < 0)
            *binary = ! isatty (STDIN_FILENO);
        if (*binary)
            xset_binary_mode (STDIN_FILENO, O_BINARY);
    }
}
else
{
    fp = fopen (filename, (O_BINARY && *binary ? "rb" : "r"));
    if (fp == nullptr)
    {
        if (ignore_missing && errno == ENOENT)
        {
            *missing = true;
            return true;
        }
        error (0, errno, "%s", quotef (filename));
        return false;
    }
}
fadvise (fp, FADVISE_SEQUENTIAL);

#if HASH_ALGO_CKSUM
if (cksum_algorithm == blake2b)
    *length = digest_length / 8;
err = DIGEST_STREAM (fp, bin_result, length);
#endif
#endif

```

```

err = DIGEST_STREAM (fp, bin_result, length);
#elif HASH_ALGO_BLAKE2
err = DIGEST_STREAM (fp, bin_result, digest_length / 8);
#else
err = DIGEST_STREAM (fp, bin_result);
#endif
err = err ? errno : 0;
if (is_stdin)
    clearerr (fp);
else if (fclose (fp) != 0 && !err)
    err = errno;

if (err)
{
    error (0, err, "%s", quotef (filename));
    return false;
}

return true;
}

#if !HASH_ALGO_SUM
static void
output_file (char const *file, int binary_file, void const *digest,
             bool raw, bool tagged, unsigned char delim, MAYBE_UNUSED bool args,
             MAYBE_UNUSED uintmax_t length)
{
# if HASH_ALGO_CKSUM
if (raw)
{
    fwrite (digest, 1, digest_length / 8, stdout);
    return;
}
# endif

unsigned char const *bin_buffer = digest;

/* Output a leading backslash if the file name contains problematic chars. */
bool needs_escape = delim == '\n' && problematic_chars (file);

if (needs_escape)
    putchar ('\\');

if (tagged)
{
    fputs (DIGEST_TYPE_STRING, stdout);
# if HASH_ALGO_BLAKE2
    if (digest_length < BLAKE2B_MAX_LEN * 8)
        printf ("-%ju", digest_length);
# elif HASH_ALGO_CKSUM
    if (cksum_algorithm == blake2b)
    {
        if (digest_length < BLAKE2B_MAX_LEN * 8)
            printf ("-%ju", digest_length);
    }
# endif
    fputs (" (", stdout);
    print_filename (file, needs_escape);
    fputs (") = ", stdout);
}

```

```

# if HASH_ALGO_CKSUM
if (base64_digest)
{
    char b64[BASE64_LENGTH (DIGEST_BIN_BYTES) + 1];
    base64_encode ((char const *) bin_buffer, digest_length / 8,
                   b64, sizeof b64);
    fputs (b64, stdout);
}
else
# endif
{
    for (size_t i = 0; i < (digest_hex_bytes / 2); ++i)
        printf ("%02x", bin_buffer[i]);
}

if (!tagged)
{
    putchar (' ');
    putchar (binary_file ? '*' : ' ');
    print_filename (file, needs_escape);
}

putchar (delim);
}
#endif

#if HASH_ALGO_CKSUM
/* Return true if B64_DIGEST is the same as the base64 digest of the
   DIGEST_LENGTH/8 bytes at BIN_BUFFER. */
static bool
b64_equal (unsigned char const *b64_digest, unsigned char const *bin_buffer)
{
    size_t b64_n_bytes = BASE64_LENGTH (digest_length / 8);
    char b64[BASE64_LENGTH (DIGEST_BIN_BYTES) + 1];
    base64_encode ((char const *) bin_buffer, digest_length / 8, b64, sizeof b64);
    return memcmp (b64_digest, b64, b64_n_bytes + 1) == 0;
}
#endif

/* Return true if HEX_DIGEST is the same as the hex-encoded digest of the
   DIGEST_LENGTH/8 bytes at BIN_BUFFER. */
static bool
hex_equal (unsigned char const *hex_digest, unsigned char const *bin_buffer)
{
    static const char bin2hex[] = { '0', '1', '2', '3',
                                   '4', '5', '6', '7',
                                   '8', '9', 'a', 'b',
                                   'c', 'd', 'e', 'f' };
    size_t digest_bin_bytes = digest_hex_bytes / 2;

    /* Compare generated binary number with text representation
       in check file. Ignore case of hex digits. */
    size_t cnt;
    for (cnt = 0; cnt < digest_bin_bytes; ++cnt)
    {
        if (c_tolower (hex_digest[2 * cnt])
            != bin2hex[bin_buffer[cnt] >> 4]
            || (c_tolower (hex_digest[2 * cnt + 1])
                != (bin2hex[bin_buffer[cnt] & 0xf])))
    }
}

```

```

        break;
    }
    return cnt == digest_bin_bytes;
}

static bool
digest_check (char const *checkfile_name)
{
FILE *checkfile_stream;
uintmax_t n_misformatted_lines = 0;
uintmax_t n_mismatched_checksums = 0;
uintmax_t n_open_or_read_failures = 0;
bool properly_formatted_lines = false;
bool matched_checksums = false;
unsigned char bin_buffer_unaligned[DIGEST_BIN_BYTES + DIGEST_ALIGN];
/* Make sure bin_buffer is properly aligned. */
unsigned char *bin_buffer = ptr_align (bin_buffer_unaligned, DIGEST_ALIGN);
uintmax_t line_number;
char *line;
size_t line_chars_allocated;
bool is_stdin = STREQ (checkfile_name, "-");

if (is_stdin)
{
    have_read_stdin = true;
    checkfile_name = _("standard input");
    checkfile_stream = stdin;
}
else
{
    checkfile_stream = fopen (checkfile_name, "r");
    if (checkfile_stream == nullptr)
    {
        error (0, errno, "%s", quotef (checkfile_name));
        return false;
    }
}
line_number = 0;
line = nullptr;
line_chars_allocated = 0;
do
{
{
    char *filename;
    int binary;
    unsigned char *digest;
    ssize_t line_length;

    ++line_number;
    if (line_number == 0)
        error (EXIT_FAILURE, 0, _("'%s: too many checksum lines"),
              quotef (checkfile_name));

    line_length = getline (&line, &line_chars_allocated, checkfile_stream);
    if (line_length <= 0)
        break;

    /* Ignore comment lines, which begin with a '#' character. */
    if (line[0] == '#')
        continue;
}
}

```

```

/* Remove any trailing newline. */
line_length -= line[line_length - 1] == '\n';
/* Remove any trailing carriage return. */
line_length -= line[line_length - (0 < line_length)] == '\r';

/* Ignore empty lines. */
if (line_length == 0)
    continue;

line[line_length] = '\0';

size_t d_len;
if (! (split_3 (line, line_length, &digest, &d_len, &binary, &filename)
      && ! (is_stdin && STREQ (filename, "-"))))
{
    ++n_misformatted_lines;

    if (warn)
    {
        error (0, 0,
               ("%s: %ju"
                ": improperly formatted %s checksum line"),
               quotef (checkfile_name), line_number,
               DIGEST_TYPE_STRING);
    }
}
else
{
    bool ok;
    bool missing;
    bool needs_escape = ! status_only && problematic_chars (filename);

    properly_formatted_lines = true;

    uintmax_t length;
    ok = digest_file (filename, &binary, bin_buffer, &missing, &length);

    if (!ok)
    {
        ++n_open_or_read_failures;
        if (!status_only)
        {
            if (needs_escape)
                putchar ('\\');
            print_filename (filename, needs_escape);
            printf ("(%s)\n", _("FAILED open or read"));
        }
    }
    else if (ignore_missing && missing)
    {
        /* Ignore missing files with --ignore-missing. */
        ;
    }
    else
    {
        bool match = false;
#if HASH_ALGO_CKSUM
        if (d_len < digest_hex_bytes)
            match = b64_equal (digest, bin_buffer);

```

```

        else
#endif
        if (d_len == digest_hex_bytes)
            match = hex_equal (digest, bin_buffer);

        if (match)
            matched_checksums = true;
        else
            ++n_mismatched_checksums;

        if (!status_only)
        {
            if (! match || ! quiet)
            {
                if (needs_escape)
                    putchar ('\\');
                print_filename (filename, needs_escape);
            }

            if (! match)
                printf (" : %s\n", _("FAILED"));
            else if (!quiet)
                printf (" : %s\n", _("OK"));
            }
        }
    }

while (!feof (checkfile_stream) && !ferror (checkfile_stream));

free (line);

int err = ferror (checkfile_stream) ? 0 : -1;
if (is_stdin)
    clearerr (checkfile_stream);
else if (fclose (checkfile_stream) != 0 && err < 0)
    err = errno;

if (0 <= err)
{
    error (0, err, err ? "%s" : _("'%s: read error"),
           quotef (checkfile_name));
    return false;
}

if (! properly_formatted_lines)
{
    /* Warn if no tests are found. */
    error (0, 0, _("'%s: no properly formatted checksum lines found"),
           quotef (checkfile_name));
}
else
{
    if (!status_only)
    {
        if (n_misformatted_lines != 0)
            error (0, 0,
                   (ngettext
                     ("WARNING: %ju line is improperly formatted",
                      "WARNING: %ju lines are improperly formatted",
                      select_plural (n_misformatted_lines))),
```

```

n_misformatted_lines);

if (n_open_or_read_failures != 0)
    error (0, 0,
        (ngettext
            ("WARNING: %ju listed file could not be read",
             "WARNING: %ju listed files could not be read",
             select_plural (n_open_or_read_failures)),
            n_open_or_read_failures);

if (n_mismatched_checksums != 0)
    error (0, 0,
        (ngettext
            ("WARNING: %ju computed checksum did NOT match",
             "WARNING: %ju computed checksums did NOT match",
             select_plural (n_mismatched_checksums)),
            n_mismatched_checksums);

if (ignore_missing && ! matched_checksums)
    error (0, 0, _("%s: no file was verified"),
           quotef (checkfile_name));
}

}

return (properly_formatted_lines
    && matched_checksums
    && n_mismatched_checksums == 0
    && n_open_or_read_failures == 0
    && (!strict || n_misformatted_lines == 0));
}

int
main (int argc, char **argv)
{
unsigned char bin_buffer_unaligned[DIGEST_BIN_BYTES + DIGEST_ALIGN];
/* Make sure bin_buffer is properly aligned. */
unsigned char *bin_buffer = ptr_align (bin_buffer_unaligned, DIGEST_ALIGN);
bool do_check = false;
int opt;
bool ok = true;
int binary = -1;
int prefix_tag = -1;

/* Setting values of global variables. */
initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

atexit (close_stdout);

/* Line buffer stdout to ensure lines are written atomically and immediately
   so that processes running in parallel do not intersperse their output. */
setvbuf (stdout, nullptr, _IOLBF, 0);

#if HASH_ALGO_SUM
char const *short_opts = "rs";
#elif HASH_ALGO_CKSUM
char const *short_opts = "a:l:bctwz";

```

```

char const *digest_length_str = "";
#ifndef HASH_ALGO_BLAKE2
char const *short_opts = "l:bctwz";
char const *digest_length_str = "";
#else
char const *short_opts = "bctwz";
#endif

while ((opt = getopt_long (argc, argv, short_opts, long_options, nullptr))
       != -1)
    switch (opt)
    {
#ifndef HASH_ALGO_CKSUM
        case 'a':
            cksum_algorithm = XARGMATCH_EXACT ("--algorithm", optarg,
                                              algorithm_args, algorithm_types);
            algorithm_specified = true;
            break;

```

- case DEBUG\_PROGRAM\_OPTION:
 cksum\_debug = true;
 break;

```

#endif
#ifndef HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
        case 'l':
            digest_length = xdectoumax (optarg, 0, UINTMAX_MAX, "",
                                         ("invalid length"), 0);
            digest_length_str = optarg;
            break;
#endif
#ifndef !HASH_ALGO_SUM
        case 'c':
            do_check = true;
            break;
        case STATUS_OPTION:
            status_only = true;
            warn = false;
            quiet = false;
            break;
        case 'b':
            binary = 1;
            break;
        case 't':
            binary = 0;
            break;
        case 'w':
            status_only = false;
            warn = true;
            quiet = false;
            break;

```

- case IGNORE\_MISSING\_OPTION:
 ignore\_missing = true;
 break;

```

        case QUIET_OPTION:
            status_only = false;
            warn = false;
            quiet = true;
            break;
        case STRICT_OPTION:
            strict = true;

```

```

        break;
# if HASH_ALGO_CKSUM
    case BASE64_OPTION:
        base64_digest = true;
        break;
    case RAW_OPTION:
        raw_digest = true;
        break;
    case UNTAG_OPTION:
        if (prefix_tag == 1)
            binary = -1;
        prefix_tag = 0;
        break;
# endif
    case TAG_OPTION:
        prefix_tag = 1;
        binary = 1;
        break;
    case 'z':
        digest_delim = '\0';
        break;
#endif
#endif HASH_ALGO_SUM
case 'r':           /* For SysV compatibility. */
    sum_algorithm = bsd;
    break;

case 's':
    sum_algorithm = sysv;
    break;
#endif
case_GETOPT_HELP_CHAR;
case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);
default:
    usage (EXIT_FAILURE);
}

min_digest_line_length = MIN_DIGEST_LINE_LENGTH;
#ifndef HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
#ifndef HASH_ALGO_CKSUM
if (digest_length && cksum_algorithm != blake2b)
    error (EXIT_FAILURE, 0,
           _("--length is only supported with --algorithm=blake2b"));
#endif
if (digest_length % 8 != 0)
{
    error (0, 0, _("invalid length: %s"), quote (digest_length_str));
    error (EXIT_FAILURE, 0, _("length is not a multiple of 8"));
}
if (digest_length > BLAKE2B_MAX_LEN * 8)
{
    error (0, 0, _("invalid length: %s"), quote (digest_length_str));
    error (EXIT_FAILURE, 0,
           _("maximum digest length for %s is %d bits"),
           quote (DIGEST_TYPE_STRING),
           BLAKE2B_MAX_LEN * 8);
}
if (digest_length == 0)
{
#endif HASH_ALGO_BLAKE2

```

```

    digest_length = BLAKE2B_MAX_LEN * 8;
# else
    digest_length = algorithm_bits[cksum_algorithm];
# endif
}
digest_hex_bytes = digest_length / 4;
#else
digest_hex_bytes = DIGEST_HEX_BYTES;
#endif

#if HASH_ALGO_CKSUM
switch (cksum_algorithm)
{
case bsd:
case sysv:
case crc:
    if (do_check && algorithm_specified)
        error (EXIT_FAILURE, 0,
               _("--check is not supported with --algorithm={bsd,sysv,crc}"));
    break;
default:
    break;
}

if (base64_digest && raw_digest)
{
    error (0, 0, _("--base64 and --raw are mutually exclusive"));
    usage (EXIT_FAILURE);
}
#endif

if (prefix_tag == -1)
    prefix_tag = HASH_ALGO_CKSUM;

if (prefix_tag && !binary)
{
    /* This could be supported in a backwards compatible way
       by prefixing the output line with a space in text mode.
       However that's invasive enough that it was agreed to
       not support this mode with --tag, as --text use cases
       are adequately supported by the default output format. */
#ifndef HASH_ALGO_CKSUM
    error (0, 0, _("--tag does not support --text mode"));
#else
    error (0, 0, _("--text mode is only supported with --untagged"));
#endif
    usage (EXIT_FAILURE);
}

if (digest_delim != '\n' && do_check)
{
    error (0, 0, _("the --zero option is not supported when "
                   "verifying checksums"));
    usage (EXIT_FAILURE);
}
#ifndef HASH_ALGO_CKSUM
if (prefix_tag && do_check)
{
    error (0, 0, _("the --tag option is meaningless when "
                   "verifying checksums"));
}

```

```

        usage (EXIT_FAILURE);
    }
#endif

if (0 <= binary && do_check)
{
    error (0, 0, _("the --binary and --text options are meaningless when "
                   "verifying checksums"));
    usage (EXIT_FAILURE);
}

if (ignore_missing && !do_check)
{
    error (0, 0,
           _("the --ignore-missing option is meaningful only when "
             "verifying checksums"));
    usage (EXIT_FAILURE);
}

if (status_only && !do_check)
{
    error (0, 0,
           _("the --status option is meaningful only when verifying checksums"));
    usage (EXIT_FAILURE);
}

if (warn && !do_check)
{
    error (0, 0,
           _("the --warn option is meaningful only when verifying checksums"));
    usage (EXIT_FAILURE);
}

if (quiet && !do_check)
{
    error (0, 0,
           _("the --quiet option is meaningful only when verifying checksums"));
    usage (EXIT_FAILURE);
}

if (strict & !do_check)
{
    error (0, 0,
           _("the --strict option is meaningful only when verifying checksums"));
    usage (EXIT_FAILURE);
}

if (!O_BINARY && binary < 0)
    binary = 0;

char **operand_lim = argv + argc;
if (optind == argc)
    *operand_lim++ = bad_cast ("-");
else if (1 < argc - optind && raw_digest)
    error (EXIT_FAILURE, 0,
           _("the --raw option is not supported with multiple files"));

for (char **operandp = argv + optind; operandp < operand_lim; operandp++)
{
    char *file = *operandp;
}

```

```

if (do_check)
    ok &= digest_check (file);
else
{
    int binary_file = binary;
    bool missing;
    uintmax_t length;

    if (! digest_file (file, &binary_file, bin_buffer, &missing, &length))
        ok = false;
    else
    {
        DIGEST_OUT (file, binary_file, bin_buffer, raw_digest, prefix_tag,
                    digest_delim, optind != argc, length);
    }
}

if (have_read_stdin && fclose (stdin) == EOF)
    error (EXIT_FAILURE, errno, _("standard input"));

return ok ? EXIT_SUCCESS : EXIT_FAILURE;
}
"""
"error_category": "File Handling and I/O Behavior on z/OS",
"error": "EDC5127I: Input/output operation failed due to incompatibility with text and binary
mode, which could be triggered by the logic involving O_BINARY or __MVS__.",
"correct_code":
"""
/* Compute checksums of files or strings.
Copyright (C) 1995-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */

/* Written by Ulrich Drepper <drepper@gnu.ai.mit.edu>. */

#include <config.h>

#include <getopt.h>
#include <sys/types.h>

#include "system.h"
#include "argmatch.h"
#include "c-ctype.h"
#include "quote.h"
#include "xdecoint.h"
#include "xstrtol.h"

#ifndef HASH_ALGO_CKSUM

```

```

#define HASH_ALGO_CKSUM 0
#endif

#if HASH_ALGO_SUM || HASH_ALGO_CKSUM
#include "sum.h"
#endif
#if HASH_ALGO_CKSUM
#include "cksum.h"
#include "base64.h"
#endif
#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
#include "blake2/b2sum.h"
#endif
#if HASH_ALGO_MD5 || HASH_ALGO_CKSUM
#include "md5.h"
#endif
#if HASH_ALGO_SHA1 || HASH_ALGO_CKSUM
#include "sha1.h"
#endif
#if HASH_ALGO_SHA256 || HASH_ALGO_SHA224 || HASH_ALGO_CKSUM
#include "sha256.h"
#endif
#if HASH_ALGO_SHA512 || HASH_ALGO_SHA384 || HASH_ALGO_CKSUM
#include "sha512.h"
#endif
#if HASH_ALGO_CKSUM
#include "sm3.h"
#endif
#include "fadvise.h"
#include "stdio--.h"
#include "xbinary-io.h"

/* The official name of this program (e.g., no 'g' prefix). */
#ifndef PROGRAM_NAME
#define PROGRAM_NAME "sum"
#define DIGEST_TYPE_STRING "BSD"
#define DIGEST_STREAM sumfns[sum_algorithm]
#define DIGEST_OUT sum_output_fns[sum_algorithm]
#define DIGEST_BITS 16
#define DIGEST_ALIGN 4
#elif HASH_ALGO_CKSUM
#define MAX_DIGEST_BITS 512
#define MAX_DIGEST_ALIGN 8
#define PROGRAM_NAME "cksum"
#define DIGEST_TYPE_STRING algorithm_tags[cksum_algorithm]
#define DIGEST_STREAM cksumfns[cksum_algorithm]
#define DIGEST_OUT cksum_output_fns[cksum_algorithm]
#define DIGEST_BITS MAX_DIGEST_BITS
#define DIGEST_ALIGN MAX_DIGEST_ALIGN
#elif HASH_ALGO_MD5
#define PROGRAM_NAME "md5sum"
#define DIGEST_TYPE_STRING "MD5"
#define DIGEST_STREAM md5_stream
#define DIGEST_BITS 128
#define DIGEST_REFERENCE "RFC 1321"
#define DIGEST_ALIGN 4
#elif HASH_ALGO_BLAKE2
#define PROGRAM_NAME "b2sum"
#define DIGEST_TYPE_STRING "BLAKE2b"
#define DIGEST_STREAM blake2b_stream

```

```

#define DIGEST_BITS 512
#define DIGEST_REFERENCE "RFC 7693"
#define DIGEST_ALIGN 8
#elif HASH_ALGO_SHA1
#define PROGRAM_NAME "sha1sum"
#define DIGEST_TYPE_STRING "SHA1"
#define DIGEST_STREAM sha1_stream
#define DIGEST_BITS 160
#define DIGEST_REFERENCE "FIPS-180-1"
#define DIGEST_ALIGN 4
#elif HASH_ALGO_SHA256
#define PROGRAM_NAME "sha256sum"
#define DIGEST_TYPE_STRING "SHA256"
#define DIGEST_STREAM sha256_stream
#define DIGEST_BITS 256
#define DIGEST_REFERENCE "FIPS-180-2"
#define DIGEST_ALIGN 4
#elif HASH_ALGO_SHA224
#define PROGRAM_NAME "sha224sum"
#define DIGEST_TYPE_STRING "SHA224"
#define DIGEST_STREAM sha224_stream
#define DIGEST_BITS 224
#define DIGEST_REFERENCE "RFC 3874"
#define DIGEST_ALIGN 4
#elif HASH_ALGO_SHA512
#define PROGRAM_NAME "sha512sum"
#define DIGEST_TYPE_STRING "SHA512"
#define DIGEST_STREAM sha512_stream
#define DIGEST_BITS 512
#define DIGEST_REFERENCE "FIPS-180-2"
#define DIGEST_ALIGN 8
#elif HASH_ALGO_SHA384
#define PROGRAM_NAME "sha384sum"
#define DIGEST_TYPE_STRING "SHA384"
#define DIGEST_STREAM sha384_stream
#define DIGEST_BITS 384
#define DIGEST_REFERENCE "FIPS-180-2"
#define DIGEST_ALIGN 8
#else
#error "Can't decide which hash algorithm to compile."
#endif
#if !HASH_ALGO_SUM && !HASH_ALGO_CKSUM
#define DIGEST_OUT output_file
#endif

#if HASH_ALGO_SUM
#define AUTHORS \
proper_name("Kayvan Aghaiepour"), \
proper_name("David MacKenzie")
#elif HASH_ALGO_CKSUM
#define AUTHORS \
proper_name_lite("Padraig Brady", "P\303\241draig Brady"), \
proper_name("Q. Frank Xia")
#elif HASH_ALGO_BLAKE2
#define AUTHORS \
proper_name_lite("Padraig Brady", "P\303\241draig Brady"), \
proper_name("Samuel Neves")
#else
#define AUTHORS \
proper_name("Ulrich Drepper"), \

```

```

proper_name ("Scott Miller"), \
proper_name ("David Madore")
#endif
#if !HASH_ALGO_BLAKE2 && !HASH_ALGO_CKSUM
#define DIGEST_HEX_BYTES (DIGEST_BITS / 4)
#endif
#define DIGEST_BIN_BYTES (DIGEST_BITS / 8)

/* The minimum length of a valid digest line. This length does
not include any newline character at the end of a line. */
#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
#define MIN_DIGEST_LINE_LENGTH 3 /* With -I 8. */
#else
#define MIN_DIGEST_LINE_LENGTH \
(DIGEST_HEX_BYTES /* length of hexadecimal message digest */ \
+ 1 /* blank */ \
+ 1 /* minimum filename length */ )
#endif

#ifndef HASH_ALGO_SUM
static void
output_file (char const *file, int binary_file, void const *digest,
             bool raw, bool tagged, unsigned char delim, bool args,
             uintmax_t length);
#endif

/* True if any of the files read were the standard input. */
static bool have_read_stdin;

/* The minimum length of a valid checksum line for the selected algorithm. */
static size_t min_digest_line_length;

/* Set to the length of a digest hex string for the selected algorithm. */
static size_t digest_hex_bytes;

/* With --check, don't generate any output.
The exit code indicates success or failure. */
static bool status_only = false;

/* With --check, print a message to standard error warning about each
improperly formatted checksum line. */
static bool warn = false;

/* With --check, ignore missing files. */
static bool ignore_missing = false;

/* With --check, suppress the "OK" printed for each verified file. */
static bool quiet = false;

/* With --check, exit with a non-zero return code if any line is
improperly formatted. */
static bool strict = false;

/* Whether a BSD reversed format checksum is detected. */
static int bsd_reversed = -1;

/* line delimiter. */
static unsigned char digest_delim = '\n';

#endif HASH_ALGO_CKSUM

```

```

/* If true, print base64-encoded digests, not hex. */
static bool base64_digest = false;
#endif

/* If true, print binary digests, not hex. */
static bool raw_digest = false;

#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
#define BLAKE2B_MAX_LEN BLAKE2B_OUTBYTES
static uintmax_t digest_length;
#endif /* HASH_ALGO_BLAKE2 */

typedef void (*digest_output_fn)(char const *, int, void const *, bool,
                                bool, unsigned char, bool, uintmax_t);

#if HASH_ALGO_SUM
enum Algorithm
{
    bsd,
    sysv,
};

static enum Algorithm sum_algorithm;
static sumfn sumfns[]=
{
    bsd_sum_stream,
    sysv_sum_stream,
};
static digest_output_fn sum_output_fns[]=
{
    output_bsd,
    output_sysv,
};
#endif

#if HASH_ALGO_CKSUM
static int
md5_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return md5_stream (stream, resstream);
}
static int
sha1_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return sha1_stream (stream, resstream);
}
static int
sha224_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return sha224_stream (stream, resstream);
}
static int
sha256_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return sha256_stream (stream, resstream);
}
static int
sha384_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
    return sha384_stream (stream, resstream);
}

```

```

static int
sha512_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
return sha512_stream (stream, resstream);
}
static int
blake2b_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
return blake2b_stream (stream, resstream, *length);
}
static int
sm3_sum_stream (FILE *stream, void *resstream, uintmax_t *length)
{
return sm3_stream (stream, resstream);
}

enum Algorithm
{
bsd,
sysv,
crc,
md5,
sha1,
sha224,
sha256,
sha384,
sha512,
blake2b,
sm3,
};

static char const *const algorithm_args[] =
{
"bsd", "sysv", "crc", "md5", "sha1", "sha224",
"sha256", "sha384", "sha512", "blake2b", "sm3", nullptr
};
static enum Algorithm const algorithm_types[] =
{
bsd, sysv, crc, md5, sha1, sha224,
sha256, sha384, sha512, blake2b, sm3,
};
ARGMATCH_VERIFY (algorithm_args, algorithm_types);

static char const *const algorithm_tags[] =
{
"BSD", "SYSV", "CRC", "MD5", "SHA1", "SHA224",
"SHA256", "SHA384", "SHA512", "BLAKE2b", "SM3", nullptr
};
static int const algorithm_bits[] =
{
16, 16, 32, 128, 160, 224,
256, 384, 512, 512, 256, 0
};

static_assert (ARRAY_CARDINALITY (algorithm_bits)
== ARRAY_CARDINALITY (algorithm_args));

static bool algorithm_specified = false;
static enum Algorithm cksum_algorithm = crc;
static sumfn cksumfns[]=

```

```

{
bsd_sum_stream,
sysv_sum_stream,
crc_sum_stream,
md5_sum_stream,
sha1_sum_stream,
sha224_sum_stream,
sha256_sum_stream,
sha384_sum_stream,
sha512_sum_stream,
blake2b_sum_stream,
sm3_sum_stream,
};

static digest_output_fn cksum_output_fns[]=
{
output_bsd,
output_sysv,
output_crc,
output_file,
output_file,
output_file,
output_file,
output_file,
output_file,
output_file,
output_file,
output_file,
};

bool cksum_debug;
#endif

/* For long options that have no equivalent short option, use a
non-character as a pseudo short option, starting with CHAR_MAX + 1. */

enum
{
IGNORE_MISSING_OPTION = CHAR_MAX + 1,
STATUS_OPTION,
QUIET_OPTION,
STRICT_OPTION,
TAG_OPTION,
UNTAG_OPTION,
DEBUG_PROGRAM_OPTION,
RAW_OPTION,
BASE64_OPTION,
};

static struct option const long_options[] =
{
#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
{ "length", required_argument, nullptr, 'l' },
#endif

#if !HASH_ALGO_SUM
{ "check", no_argument, nullptr, 'c' },
{ "ignore-missing", no_argument, nullptr, IGNORE_MISSING_OPTION },
{ "quiet", no_argument, nullptr, QUIET_OPTION },
{ "status", no_argument, nullptr, STATUS_OPTION },
{ "warn", no_argument, nullptr, 'w' },
{ "strict", no_argument, nullptr, STRICT_OPTION },
{ "tag", no_argument, nullptr, TAG_OPTION },

```

```

{ "zero", no_argument, nullptr, 'z' },

# if HASH_ALGO_CKSUM
{ "algorithm", required_argument, nullptr, 'a'},
{ "base64", no_argument, nullptr, BASE64_OPTION },
{ "debug", no_argument, nullptr, DEBUG_PROGRAM_OPTION},
{ "raw", no_argument, nullptr, RAW_OPTION},
{ "untagged", no_argument, nullptr, UNTAG_OPTION },
#endif
{ "binary", no_argument, nullptr, 'b' },
{ "text", no_argument, nullptr, 't' },

#else
{"sysv", no_argument, nullptr, 's'},
#endif

{ GETOPT_HELP_OPTION_DECL },
{ GETOPT_VERSION_OPTION_DECL },
{ nullptr, 0, nullptr, 0 }
};

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    printf (_("\
Usage: %s [OPTION]... [FILE]...\n\
"), program_name);
#endif HASH_ALGO_CKSUM
    fputs (_("\
Print or verify checksums.\n\
By default use the 32 bit CRC algorithm.\n\
"), stdout);
#else
    printf (_("\
Print or check %s (%d-bit) checksums.\n\
"),
            DIGEST_TYPE_STRING,
            DIGEST_BITS);
#endif
    emit_stdin_note ();
#endif HASH_ALGO_SUM
    fputs (_("\
\n\
-r      use BSD sum algorithm (the default), use 1K blocks\n\
-s, --sysv   use System V sum algorithm, use 512 bytes blocks\n\
"), stdout);
#endif
#endif HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
    emit_mandatory_arg_note ();
#endif
#endif HASH_ALGO_CKSUM
    fputs (_("\
-a, --algorithm=TYPE select the digest type to use. See DIGEST below.\n\
\n\
"), stdout);

```

```

        fputs (_("\
--base64      emit base64-encoded digests, not hexadecimal\
\n\
"), stdout);
#endif
#if !HASH_ALGO_SUM
#if !HASH_ALGO_CKSUM
if (O_BINARY || __MVS__)
    fputs (_("\
-b, --binary   read in binary mode (default unless reading tty stdin) \
\n\
"), stdout);
else
    fputs (_("\
-b, --binary   read in binary mode\n\
"), stdout);
#endif
fputs (_("\
-c, --check    read checksums from the FILEs and check them\n\
"), stdout);
#endif HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
fputs (_("\
-l, --length=BITS digest length in bits; must not exceed the max for\n\
the blake2 algorithm and must be a multiple of 8\n\
"), stdout);
#endif
#if HASH_ALGO_CKSUM
fputs (_("\
--raw         emit a raw binary digest, not hexadecimal\
\n\
"), stdout);
fputs (_("\
--tag         create a BSD-style checksum (the default)\n\
"), stdout);
fputs (_("\
--untagged   create a reversed style checksum, without digest type\n\
"), stdout);
#endif
#else
fputs (_("\
--tag         create a BSD-style checksum\n\
"), stdout);
#endif
#endif !HASH_ALGO_CKSUM
if (O_BINARY || __MVS__)
    fputs (_("\
-t, --text     read in text mode (default if reading tty stdin)\n\
"), stdout);
else
    fputs (_("\
-t, --text     read in text mode (default)\n\
"), stdout);
#endif
-f, --force    force overwriting existing output files\n\
-z, --zero    end each output line with NUL, not newline,\n\
and disable file name escaping\n\
"), stdout);
fputs (_("\
\n\
The following five options are useful only when verifying checksums:\n\
--ignore-missing don't fail or report status for missing files\n\

```

```

--quiet      don't print OK for each successfully verified file\n\
--status     don't output anything, status code shows success\n\
--strict     exit non-zero for improperly formatted checksum lines\n\
-w, --warn   warn about improperly formatted checksum lines\n\
\n\
"), stdout);
#endif
#if HASH_ALGO_CKSUM
    fputs (_("\
        --debug      indicate which implementation used\n\
"), stdout);
#endif
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
#endif HASH_ALGO_CKSUM
    fputs (_("\
\n\
DIGEST determines the digest algorithm and default output format:\n\
sysv    (equivalent to sum -s)\n\
bsd     (equivalent to sum -r)\n\
crc     (equivalent to cksum)\n\
md5     (equivalent to md5sum)\n\
sha1    (equivalent to sha1sum)\n\
sha224   (equivalent to sha224sum)\n\
sha256   (equivalent to sha256sum)\n\
sha384   (equivalent to sha384sum)\n\
sha512   (equivalent to sha512sum)\n\
blake2b  (equivalent to b2sum)\n\
sm3     (only available through cksum)\n\
\n\
"), stdout);
#endif
#endif !HASH_ALGO_SUM && !HASH_ALGO_CKSUM
    printf (_("\
\n\
The sums are computed as described in %s.\n"), DIGEST_REFERENCE);
    fputs (_("\
When checking, the input should be a former output of this program.\n\
The default mode is to print a line with: checksum, a space,\n\
a character indicating input mode ('*' for binary, ' ' for text\n\
or where binary is insignificant), and name for each FILE.\n\
\n\
There is no difference between binary mode and text mode on GNU systems.\n\
\n\
"), stdout);
#endif
#endif HASH_ALGO_CKSUM
    fputs (_("\
When checking, the input should be a former output of this program,\n\
or equivalent standalone program.\n\
\n\
"), stdout);
#endif
    emit_ancillary_info (PROGRAM_NAME);
}

exit (status);
}

/* Given a string S, return TRUE if it contains problematic characters
 * that need escaping. Note we escape '\' itself to provide some forward
 * compat to introduce escaping of other characters. */

```

```

ATTRIBUTE_PURE
static bool
problematic_chars (char const *s)
{
size_t length = strcspn (s, "\\\n\r");
return s[length] != '\0';
}

#define ISWHITE(c) ((c) == ' ' || (c) == '\t')

/* Given a file name, S of length S_LEN, that is not NUL-terminated,
modify it in place, performing the equivalent of this sed substitution:
's\\n\\n/g;s\\\\r/r/g;s\\\\\\\\Vg' i.e., replacing each "\\n" string
with a newline, each "\\r" string with a carriage return,
and each "\\\\" with a single backslash, NUL-terminate it and return S.
If S is not a valid escaped file name, i.e., if it ends with an odd number
of backslashes or if it contains a backslash followed by anything other
than "n" or another backslash, return nullptr. */

static char *
filename_unescape (char *s, size_t s_len)
{
char *dst = s;

for (size_t i = 0; i < s_len; i++)
{
switch (s[i])
{
case '\\':
if (i == s_len - 1)
{
/* File name ends with an unescaped backslash: invalid. */
return nullptr;
}
++i;
switch (s[i])
{
case 'n':
*dst++ = '\n';
break;
case 'r':
*dst++ = '\r';
break;
case '\\':
*dst++ = '\\';
break;
default:
/* Only '\\', 'n' or 'r' may follow a backslash. */
return nullptr;
}
break;

case '\0':
/* The file name may not contain a NUL. */
return nullptr;

default:
*dst++ = s[i];
break;
}
}

```

```

        }
    if (dst < s + s_len)
        *dst = '\0';

    return s;
}

/* Return true if S is a LEN-byte NUL-terminated string of hex or base64
   digits and has the expected length. Otherwise, return false. */
ATTRIBUTE_PURE
static bool
valid_digits (unsigned char const *s, size_t len)
{
#ifndef HASH_ALGO_CKSUM
if (len == BASE64_LENGTH (digest_length / 8))
{
    size_t i;
    for (i = 0; i < len - digest_length % 3; i++)
    {
        if (!isbase64 (*s))
            return false;
        ++s;
    }
    for ( ; i < len; i++)
    {
        if (*s != '=')
            return false;
        ++s;
    }
}
else
#endif
if (len == digest_hex_bytes)
{
    for (idx_t i = 0; i < digest_hex_bytes; i++)
    {
        if (!c_isxdigit (*s))
            return false;
        ++s;
    }
}
else
    return false;

return *s == '\0';
}

/* Split the checksum string S (of length S_LEN) from a BSD 'md5' or
   'sha1' command into two parts: a hexadecimal digest, and the file
   name. S is modified. Set *D_LEN to the length of the digest string.
   Return true if successful. */
static bool
bsd_split_3 (char *s, size_t s_len,
             unsigned char **digest, size_t *d_len,
             char **file_name, bool escaped_filename)
{
if (s_len == 0)
    return false;

```

```

/* Find end of filename. */
size_t i = s_len - 1;
while (i && s[i] != ')')
    i--;
if (s[i] != ')')
    return false;

*file_name = s;

if (escaped_filename && filename_unescape (s, i) == nullptr)
    return false;

s[i++] = '\0';

while (ISWHITE (s[i]))
    i++;

if (s[i] != '=')
    return false;

i++;

while (ISWHITE (s[i]))
    i++;

*digest = (unsigned char *) &s[i];

*d_len = s_len - i;
return valid_digits (*digest, *d_len);
}

#if HASH_ALGO_CKSUM
/* Return the corresponding Algorithm for the string S,
or -1 for no match. */

static ptrdiff_t
algorithm_from_tag (char *s)
{
/* Limit check size to this length for perf reasons. */
static size_t max_tag_len;
if (!max_tag_len)
{
    char const * const * tag = algorithm_tags;
    while (*tag)
    {
        size_t tag_len = strlen (*tag++);
        max_tag_len = MAX (tag_len, max_tag_len);
    }
}
size_t i = 0;

/* Find end of tag */
while (i <= max_tag_len && s[i] && !ISWHITE (s[i])
    && s[i] != '-' && s[i] != '(')
    ++i;

if (i > max_tag_len)
    return -1;

```

```

/* Terminate tag, and lookup. */
char sep = s[i];
s[i] = '\0';
ptrdiff_t algo = argmatch_exact (s, algorithm_tags);
s[i] = sep;

return algo;
}
#endif

/* Split the string S (of length S_LEN) into three parts:
a hexadecimal digest, binary flag, and the file name.
S is modified. Set *D_LEN to the length of the digest string.
Return true if successful. */

static bool
split_3 (char *s, size_t s_len,
         unsigned char **digest, size_t *d_len, int *binary, char **file_name)
{
bool escaped_filename = false;
size_t algo_name_len;

size_t i = 0;
while (ISWHITE (s[i]))
    ++i;

if (s[i] == '\\')
{
    ++
    escaped_filename = true;
}

/* Check for BSD-style checksum line. */

#if HASH_ALGO_CKSUM
if (!algorithm_specified)
{
    ptrdiff_t algo_tag = algorithm_from_tag (s + i);
    if (algo_tag >= 0)
    {
        if (algo_tag <= crc)
            return false; /* We don't support checking these older formats. */
        cksum_algorithm = algo_tag;
    }
    else
        return false; /* We only support tagged format without -a. */
}
#endif

algo_name_len = strlen (DIGEST_TYPE_STRING);
if (STREQ_LEN (s + i, DIGEST_TYPE_STRING, algo_name_len))
{
    i += algo_name_len;
#endif HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
/* Terminate and match algorithm name. */
char const *algo_name = &s[i - algo_name_len];
bool length_specified = s[i] == '-';
bool openssl_format = s[i] == '('; /* and no length_specified */
s[i++] = '\0';

```

```

if (!STREQ (algo_name, DIGEST_TYPE_STRING))
    return false;
if (openssl_format)
    s[--i] = '(';

# if HASH_ALGO_BLAKE2
digest_length = BLAKE2B_MAX_LEN * 8;
# else
digest_length = algorithm_bits[cksum_algorithm];
# endif
if (length_specified)
{
    uintmax_t length;
    char *siend;
    if (! (xstrtoimax (s + i, &siend, 0, &length, nullptr) == LONGINT_OK
        && 0 < length && length <= digest_length
        && length % 8 == 0))
        return false;

    i = siend - s;
    digest_length = length;
}
digest_hex_bytes = digest_length / 4;
#endif
if (s[i] == ' ')
    ++i;
if (s[i] == '(')
{
    ++
    *binary = 0;
    return bsd_split_3 (s + i, s_len - i,
        digest, d_len, file_name, escaped_filename);
}
return false;
}

/* Ignore this line if it is too short.
   Each line must have at least 'min_digest_line_length - 1' (or one more, if
   the first is a backslash) more characters to contain correct message digest
   information. */
if (s_len - i < min_digest_line_length + (s[i] == '\\'))
    return false;

*digest = (unsigned char *) &s[i];

#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
/* Auto determine length. */
# if HASH_ALGO_CKSUM
if (cksum_algorithm == blake2b) {
# endif
unsigned char const *hp = *digest;
digest_hex_bytes = 0;
while (c_isxdigit (*hp++))
    digest_hex_bytes++;
if (digest_hex_bytes < 2 || digest_hex_bytes % 2
    || BLAKE2B_MAX_LEN * 2 < digest_hex_bytes)
    return false;
digest_length = digest_hex_bytes * 4;
# if HASH_ALGO_CKSUM
}

```

```

#endif
#endif

/* This field must be the hexadecimal or base64 representation
   of the message digest. */
while (s[i] && !ISWHITE (s[i]))
    i++;

/* The digest must be followed by at least one whitespace character. */
if (i == s_len)
    return false;

*d_len = &s[i] - (char *) *digest;
s[i++] = '\0';

if (! valid_digits (*digest, *d_len))
    return false;

/* If "bsd reversed" format detected. */
if ((s_len - i == 1) || (s[i] != ' ' && s[i] != '*'))
{
    /* Don't allow mixing bsd and standard formats,
       to minimize security issues with attackers
       renaming files with leading spaces.
       This assumes that with bsd format checksums
       that the first file name does not have
       a leading ' ' or '*'. */
    if (bsd_reversed == 0)
        return false;
    bsd_reversed = 1;
}
else if (bsd_reversed != 1)
{
    bsd_reversed = 0;
    *binary = (s[i++] == '*');
}

/* All characters between the type indicator and end of line are
   significant -- that includes leading and trailing white space. */
*file_name = &s[i];

if (escaped_filename)
    return filename_unescape (&s[i], s_len - i) != nullptr;

return true;
}

/* If ESCAPE is true, then translate each:
   NEWLINE byte to the string, "\\n",
   CARRIAGE RETURN byte to the string, "\\r",
   and each backslash to "\\\\". */
static void
print_filename (char const *file, bool escape)
{
if (! escape)
{
    fputs (file, stdout);
    return;
}

```

```

while (*file)
{
    switch (*file)
    {
        case '\n':
            fputs ("\n", stdout);
            break;

        case '\r':
            fputs ("\r", stdout);
            break;

        case '\\':
            fputs ("\\", stdout);
            break;

        default:
            putchar (*file);
            break;
    }
    file++;
}
}

/* An interface to the function, DIGEST_STREAM.
Operate on FILENAME (it may be "-").

*BINARY indicates whether the file is binary. BINARY < 0 means it
depends on whether binary mode makes any difference and the file is
a terminal; in that case, clear *BINARY if the file was treated as
text because it was a terminal.

```

Put the checksum in \*BIN\_RESULT, which must be properly aligned.  
Put true in \*MISSING if the file can't be opened due to ENOENT.  
Return true if successful. \*/

```

static bool
digest_file (char const *filename, int *binary, unsigned char *bin_result,
            bool *missing, MAYBE_UNUSED uintmax_t *length)
{
FILE *fp;
int err;
bool is_stdin = STREQ (filename, "-");

*missing = false;

if (is_stdin)
{
    have_read_stdin = true;
    fp = stdin;
    if ((O_BINARY || __MVS__) && *binary)
    {
        if (*binary < 0)
            *binary = ! isatty (STDIN_FILENO);
        if (*binary)
            xset_binary_mode (STDIN_FILENO, O_BINARY);
    }
}
else
{

```

```

fp = fopen (filename, ((O_BINARY || __MVS__) && *binary ? "rb" : "r"));
if (fp == nullptr)
{
    if (ignore_missing && errno == ENOENT)
    {
        *missing = true;
        return true;
    }
    error (0, errno, "%s", quotef (filename));
    return false;
}
}

#ifndef __MVS__
if (*binary)
    __disableautocvt(fileno(fp));
#endif

fadvise (fp, FADVISE_SEQUENTIAL);

#if HASH_ALGO_CKSUM
if (cksum_algorithm == blake2b)
    *length = digest_length / 8;
err = DIGEST_STREAM (fp, bin_result, length);
#elif HASH_ALGO_SUM
err = DIGEST_STREAM (fp, bin_result, length);
#elif HASH_ALGO_BLAKE2
err = DIGEST_STREAM (fp, bin_result, digest_length / 8);
#else
err = DIGEST_STREAM (fp, bin_result);
#endif
err = err ? errno : 0;
if (is_stdin)
    clearerr (fp);
else if (fclose (fp) != 0 && !err)
    err = errno;

if (err)
{
    error (0, err, "%s", quotef (filename));
    return false;
}

return true;
}

#if !HASH_ALGO_SUM
static void
output_file (char const *file, int binary_file, void const *digest,
            bool raw, bool tagged, unsigned char delim, MAYBE_UNUSED bool args,
            MAYBE_UNUSED uintmax_t length)
{
# if HASH_ALGO_CKSUM
if (raw)
{
    fwrite (digest, 1, digest_length / 8, stdout);
    return;
}
#endif

```

```

unsigned char const *bin_buffer = digest;

/* Output a leading backslash if the file name contains problematic chars. */
bool needs_escape = delim == '\n' && problematic_chars (file);

if (needs_escape)
    putchar ('\\');

if (tagged)
{
    fputs (DIGEST_TYPE_STRING, stdout);
# if HASH_ALGO_BLAKE2
    if (digest_length < BLAKE2B_MAX_LEN * 8)
        printf ("%ju", digest_length);
# elif HASH_ALGO_CRC32C
    if (cksum_algorithm == blake2b)
    {
        if (digest_length < BLAKE2B_MAX_LEN * 8)
            printf ("%ju", digest_length);
    }
# endif
    fputs (" (", stdout);
    print_filename (file, needs_escape);
    fputs (") = ", stdout);
}

# if HASH_ALGO_CRC32C
if (base64_digest)
{
    char b64[BASE64_LENGTH (DIGEST_BIN_BYTES) + 1];
    base64_encode ((char const *) bin_buffer, digest_length / 8,
                   b64, sizeof b64);
    fputs (b64, stdout);
}
else
# endif
{
    for (size_t i = 0; i < (digest_hex_bytes / 2); ++i)
        printf ("%02x", bin_buffer[i]);
}

if (!tagged)
{
    putchar (' ');
    putchar (binary_file ? '*' : ' ');
    print_filename (file, needs_escape);
}

putchar (delim);
}
#endif

#if HASH_ALGO_CRC32C
/* Return true if B64_DIGEST is the same as the base64 digest of the
   DIGEST_LENGTH/8 bytes at BIN_BUFFER. */
static bool
b64_equal (unsigned char const *b64_digest, unsigned char const *bin_buffer)
{
    size_t b64_n_bytes = BASE64_LENGTH (digest_length / 8);
    char b64[BASE64_LENGTH (DIGEST_BIN_BYTES) + 1];

```

```

base64_encode ((char const *) bin_buffer, digest_length / 8, b64, sizeof b64);
return memcmp (b64_digest, b64, b64_n_bytes + 1) == 0;
}

#endif

/* Return true if HEX_DIGEST is the same as the hex-encoded digest of the
DIGEST_LENGTH/8 bytes at BIN_BUFFER. */
static bool
hex_equal (unsigned char const *hex_digest, unsigned char const *bin_buffer)
{
    static const char bin2hex[] = { '0', '1', '2', '3',
        '4', '5', '6', '7',
        '8', '9', 'a', 'b',
        'c', 'd', 'e', 'f' };
    size_t digest_bin_bytes = digest_hex_bytes / 2;

    /* Compare generated binary number with text representation
       in check file. Ignore case of hex digits. */
    size_t cnt;
    for (cnt = 0; cnt < digest_bin_bytes; ++cnt)
    {
        if (c_tolower (hex_digest[2 * cnt])
            != bin2hex[bin_buffer[cnt] >> 4]
            || (c_tolower (hex_digest[2 * cnt + 1])
                != (bin2hex[bin_buffer[cnt] & 0xf])))
            break;
    }
    return cnt == digest_bin_bytes;
}

static bool
digest_check (char const *checkfile_name)
{
    FILE *checkfile_stream;
    uintmax_t n_misformatted_lines = 0;
    uintmax_t n_mismatched_checksums = 0;
    uintmax_t n_open_or_read_failures = 0;
    bool properly_formatted_lines = false;
    bool matched_checksums = false;
    unsigned char bin_buffer_unaligned[DIGEST_BIN_BYTES + DIGEST_ALIGN];
    /* Make sure bin_buffer is properly aligned. */
    unsigned char *bin_buffer = ptr_align (bin_buffer_unaligned, DIGEST_ALIGN);
    uintmax_t line_number;
    char *line;
    size_t line_chars_allocated;
    bool is_stdin = STREQ (checkfile_name, "-");

    if (is_stdin)
    {
        have_read_stdin = true;
        checkfile_name = _("standard input");
        checkfile_stream = stdin;
    }
    else
    {
        checkfile_stream = fopen (checkfile_name, "r");
        if (checkfile_stream == nullptr)
        {
            error (0, errno, "%s", quotef (checkfile_name));
            return false;
        }
    }
}

```

```

    }

line_number = 0;
line = nullptr;
line_chars_allocated = 0;
do
{
    char *filename;
    int binary;
    unsigned char *digest;
    ssize_t line_length;

    ++line_number;
    if (line_number == 0)
        error (EXIT_FAILURE, 0, _(""%s: too many checksum lines"),
               quotef (checkfile_name));

    line_length = getline (&line, &line_chars_allocated, checkfile_stream);
    if (line_length <= 0)
        break;

    /* Ignore comment lines, which begin with a '#' character. */
    if (line[0] == '#')
        continue;

    /* Remove any trailing newline. */
    line_length -= line[line_length - 1] == '\n';
    /* Remove any trailing carriage return. */
    line_length -= line[line_length - (0 < line_length)] == '\r';

    /* Ignore empty lines. */
    if (line_length == 0)
        continue;

    line[line_length] = '\0';

    size_t d_len;
    if (! (split_3 (line, line_length, &digest, &d_len, &binary, &filename)
          && ! (is_stdin && STREQ (filename, "-"))))
    {
        ++n_misformatted_lines;

        if (warn)
        {
            error (0, 0,
                   _(""%s: %ju
                      : improperly formatted %s checksum line"),
                   quotef (checkfile_name), line_number,
                   DIGEST_TYPE_STRING);
        }
    }
else
{
    bool ok;
    bool missing;
    bool needs_escape = ! status_only && problematic_chars (filename);

    properly_formatted_lines = true;
}
}

```

```

        uintmax_t length;
        ok = digest_file (filename, &binary, bin_buffer, &missing, &length);

        if (!ok)
        {
            ++n_open_or_read_failures;
            if (!status_only)
            {
                if (needs_escape)
                    putchar ('\\');
                print_filename (filename, needs_escape);
                printf (" : %s\n", _("FAILED open or read"));
            }
        }
        else if (ignore_missing && missing)
        {
            /* Ignore missing files with --ignore-missing. */
            ;
        }
        else
        {
            bool match = false;
#ifndef HASH_ALGO_CKSUM
            if (d_len < digest_hex_bytes)
                match = b64_equal (digest, bin_buffer);
            else
#endif
                if (d_len == digest_hex_bytes)
                    match = hex_equal (digest, bin_buffer);

            if (match)
                matched_checksums = true;
            else
                ++n_mismatched_checksums;

            if (!status_only)
            {
                if (!match || !quiet)
                {
                    if (needs_escape)
                        putchar ('\\');
                    print_filename (filename, needs_escape);
                }

                if (!match)
                    printf (" : %s\n", _("FAILED"));
                else if (!quiet)
                    printf (" : %s\n", _("OK"));
                }
            }
        }
    }

while (!feof (checkfile_stream) && !ferror (checkfile_stream));

free (line);

int err = ferror (checkfile_stream) ? 0 : -1;
if (is_stdin)
    clearerr (checkfile_stream);
else if (fclose (checkfile_stream) != 0 && err < 0)

```

```

err = errno;

if (0 <= err)
{
    error (0, err, err ? "%s" : _("'%s: read error"),
           quotef (checkfile_name));
    return false;
}

if (!properly_formatted_lines)
{
    /* Warn if no tests are found. */
    error (0, 0, _("'%s: no properly formatted checksum lines found"),
           quotef (checkfile_name));
}
else
{
    if (!status_only)
    {
        if (n_misformatted_lines != 0)
            error (0, 0,
                   (ngettext
                     ("WARNING: %ju line is improperly formatted",
                      "WARNING: %ju lines are improperly formatted",
                      select_plural (n_misformatted_lines)),
                     n_misformatted_lines);

        if (n_open_or_read_failures != 0)
            error (0, 0,
                   (ngettext
                     ("WARNING: %ju listed file could not be read",
                      "WARNING: %ju listed files could not be read",
                      select_plural (n_open_or_read_failures)),
                     n_open_or_read_failures);

        if (n_mismatched_checksums != 0)
            error (0, 0,
                   (ngettext
                     ("WARNING: %ju computed checksum did NOT match",
                      "WARNING: %ju computed checksums did NOT match",
                      select_plural (n_mismatched_checksums)),
                     n_mismatched_checksums);

        if (ignore_missing && !matched_checksums)
            error (0, 0, _("'%s: no file was verified"),
                   quotef (checkfile_name));
    }
}

return (properly_formatted_lines
&& matched_checksums
&& n_mismatched_checksums == 0
&& n_open_or_read_failures == 0
&& (!strict || n_misformatted_lines == 0));
}

int
main (int argc, char **argv)
{
    unsigned char bin_buffer_unaligned[DIGEST_BIN_BYTES + DIGEST_ALIGN];

```

```

/* Make sure bin_buffer is properly aligned. */
unsigned char *bin_buffer = ptr_align (bin_buffer_unaligned, DIGEST_ALIGN);
bool do_check = false;
int opt;
bool ok = true;
int binary = -1;
int prefix_tag = -1;

/* Setting values of global variables. */
initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

atexit (close_stdout);

/* Line buffer stdout to ensure lines are written atomically and immediately
   so that processes running in parallel do not intersperse their output. */
setvbuf (stdout, nullptr, _IOLBF, 0);

#if HASH_ALGO_SUM
char const *short_opts = "rs";
#elif HASH_ALGO_CKSUM
char const *short_opts = "a:l:bctwz";
char const *digest_length_str = "";
#elif HASH_ALGO_BLAKE2
char const *short_opts = "l:bctwz";
char const *digest_length_str = "";
#else
char const *short_opts = "bctwz";
#endif

while ((opt = getopt_long (argc, argv, short_opts, long_options, nullptr))
       != -1)
    switch (opt)
    {
#if HASH_ALGO_CKSUM
        case 'a':
            cksum_algorithm = XARGMATCH_EXACT ("--algorithm", optarg,
                                              algorithm_args, algorithm_types);
            algorithm_specified = true;
            break;
#endif
        case DEBUG_PROGRAM_OPTION:
            cksum_debug = true;
            break;
#endif
#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
        case 'l':
            digest_length = xdectoumax (optarg, 0, UINTMAX_MAX, "",
                                       ("invalid length"), 0);
            digest_length_str = optarg;
            break;
#endif
#if !HASH_ALGO_SUM
        case 'c':
            do_check = true;
            break;
        case STATUS_OPTION:

```

```

status_only = true;
warn = false;
quiet = false;
break;
case 'b':
    binary = 1;
    break;
case 't':
    binary = 0;
    break;
case 'w':
    status_only = false;
    warn = true;
    quiet = false;
    break;
case IGNORE_MISSING_OPTION:
    ignore_missing = true;
    break;
case QUIET_OPTION:
    status_only = false;
    warn = false;
    quiet = true;
    break;
case STRICT_OPTION:
    strict = true;
    break;
# if HASH_ALGO_CKSUM
case BASE64_OPTION:
    base64_digest = true;
    break;
case RAW_OPTION:
    raw_digest = true;
    break;
case UNTAG_OPTION:
    if (prefix_tag == 1)
        binary = -1;
    prefix_tag = 0;
    break;
#endif
case TAG_OPTION:
    prefix_tag = 1;
    binary = 1;
    break;
case 'z':
    digest_delim = '\0';
    break;
#endif
#endif HASH_ALGO_SUM
case 'r':          /* For SysV compatibility. */
    sum_algorithm = bsd;
    break;

case 's':
    sum_algorithm = sysv;
    break;
#endif
case_GETOPT_HELP_CHAR;
case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);
default:
    usage (EXIT_FAILURE);

```

```

}

min_digest_line_length = MIN_DIGEST_LINE_LENGTH;
#if HASH_ALGO_BLAKE2 || HASH_ALGO_CKSUM
# if HASH_ALGO_CKSUM
if (digest_length && cksum_algorithm != blake2b)
    error (EXIT_FAILURE, 0,
        _("--length is only supported with --algorithm=blake2b"));
#endif
if (digest_length % 8 != 0)
{
    error (0, 0, _("invalid length: %s"), quote (digest_length_str));
    error (EXIT_FAILURE, 0, _("length is not a multiple of 8"));
}
if (digest_length > BLAKE2B_MAX_LEN * 8)
{
    error (0, 0, _("invalid length: %s"), quote (digest_length_str));
    error (EXIT_FAILURE, 0,
        _("maximum digest length for %s is %d bits"),
        quote (DIGEST_TYPE_STRING),
        BLAKE2B_MAX_LEN * 8);
}
if (digest_length == 0)
{
# if HASH_ALGO_BLAKE2
    digest_length = BLAKE2B_MAX_LEN * 8;
# else
    digest_length = algorithm_bits[cksum_algorithm];
#endif
}
digest_hex_bytes = digest_length / 4;
#else
digest_hex_bytes = DIGEST_HEX_BYTES;
#endif

#endif HASH_ALGO_CKSUM
switch (cksum_algorithm)
{
case bsd:
case sysv:
case crc:
    if (do_check && algorithm_specified)
        error (EXIT_FAILURE, 0,
            _("--check is not supported with --algorithm={bsd,sysv,crc}"));
    break;
default:
    break;
}

if (base64_digest && raw_digest)
{
    error (0, 0, _("--base64 and --raw are mutually exclusive"));
    usage (EXIT_FAILURE);
}
#endif

if (prefix_tag == -1)
    prefix_tag = HASH_ALGO_CKSUM;

if (prefix_tag && !binary)

```

```

{
    /* This could be supported in a backwards compatible way
       by prefixing the output line with a space in text mode.
       However that's invasive enough that it was agreed to
       not support this mode with --tag, as --text use cases
       are adequately supported by the default output format. */
#endif !HASH_ALGO_CKSUM
    error (0, 0, _("--tag does not support --text mode"));
#else
    error (0, 0, _("--text mode is only supported with --untagged"));
#endif
    usage (EXIT_FAILURE);
}

if (digest_delim != '\n' && do_check)
{
    error (0, 0, _("the --zero option is not supported when "
                   "verifying checksums"));
    usage (EXIT_FAILURE);
}
#endif !HASH_ALGO_CKSUM
if (prefix_tag && do_check)
{
    error (0, 0, _("the --tag option is meaningless when "
                   "verifying checksums"));
    usage (EXIT_FAILURE);
}
#endif

if (0 <= binary && do_check)
{
    error (0, 0, _("the --binary and --text options are meaningless when "
                   "verifying checksums"));
    usage (EXIT_FAILURE);
}

if (ignore_missing && !do_check)
{
    error (0, 0,
           _("the --ignore-missing option is meaningful only when "
             "verifying checksums"));
    usage (EXIT_FAILURE);
}

if (status_only && !do_check)
{
    error (0, 0,
           _("the --status option is meaningful only when verifying checksums"));
    usage (EXIT_FAILURE);
}

if (warn && !do_check)
{
    error (0, 0,
           _("the --warn option is meaningful only when verifying checksums"));
    usage (EXIT_FAILURE);
}

if (quiet && !do_check)
{

```

```

error (0, 0,
      _("the --quiet option is meaningful only when verifying checksums"));
usage (EXIT_FAILURE);
}

if (strict & !do_check)
{
    error (0, 0,
          _("the --strict option is meaningful only when verifying checksums"));
    usage (EXIT_FAILURE);
}

if (!(O_BINARY || __MVS__)
    && binary < 0)
    binary = 0;

char **operand_lim = argv + argc;
if (optind == argc)
    *operand_lim++ = bad_cast ("-");
else if (1 < argc - optind && raw_digest)
    error (EXIT_FAILURE, 0,
          _("the --raw option is not supported with multiple files"));

for (char **operandp = argv + optind; operandp < operand_lim; operandp++)
{
    char *file = *operandp;
    if (do_check)
        ok &= digest_check (file);
    else
    {
        int binary_file = binary;
        bool missing;
        uintmax_t length;

        if (! digest_file (file, &binary_file, bin_buffer, &missing, &length))
            ok = false;
        else
        {
            DIGEST_OUT (file, binary_file, bin_buffer, raw_digest, prefix_tag,
                        digest_delim, optind != argc, length);
        }
    }
}

if (have_read_stdin && fclose (stdin) == EOF)
    error (EXIT_FAILURE, errno, _("standard input"));

return ok ? EXIT_SUCCESS : EXIT_FAILURE;
}
""",
"patch" :
diff --git a/src/digest.c b/src/digest.c
index 3d239da..22e5a4b 100644
--- a/src/digest.c
+++ b/src/digest.c
@@ -451,7 +451,7 @@ Print or check %s (%d-bit) checksums.\n\
#endif
#if !HASH_ALGO_SUM
# if !HASH_ALGO_CKSUM
-    if (O_BINARY)

```

```

+    if (O_BINARY || __MVS__)
+        fputs (_("\
-b, --binary      read in binary mode (default unless reading tty stdin)\\
\n\
@@ -487,7 +487,7 @@ Print or check %s (%d-bit) checksums.\n\
"), stdout);
# endif
# if !HASH_ALGO_CKSUM
-    if (O_BINARY)
+    if (O_BINARY || __MVS__)
        fputs (_("\
-t, --text      read in text mode (default if reading tty stdin)\n\
"), stdout);
@@ -974,7 +974,7 @@ digest_file (char const *filename, int *binary, unsigned char
*bin_result,
{
    have_read_stdin = true;
    fp = stdin;
-    if (O_BINARY && *binary)
+    if ((O_BINARY || __MVS__) && *binary)
    {
        if (*binary < 0)
            *binary = !isatty (STDIN_FILENO);
@@ -984,7 +984,7 @@ digest_file (char const *filename, int *binary, unsigned char
*bin_result,
}
else
{
-    fp = fopen (filename, (O_BINARY && *binary ? "rb" : "r"));
+    fp = fopen (filename, ((O_BINARY || __MVS__) && *binary ? "rb" : "r"));
    if (fp == nullptr)
    {
        if (ignore_missing && errno == ENOENT)
@@ -997,6 +997,11 @@ digest_file (char const *filename, int *binary, unsigned char
*bin_result,
}
}

+#ifdef __MVS__
+    if (*binary)
+        __disableautocvt(fileno(fp));
#endif
+
fadvise (fp, FADVISE_SEQUENTIAL);

#if HASH_ALGO_CKSUM
@@ -1590,7 +1595,7 @@ main (int argc, char **argv)
    usage (EXIT_FAILURE);
}

- if (!O_BINARY && binary < 0)
+ if (!(O_BINARY || __MVS__) && binary < 0)
    binary = 0;

    char **operand_lim = argv + argc;
    """
},
{
    "wrong_code":
```

/\* 'dir', 'vdir' and 'ls' directory listing programs for GNU.  
Copyright (C) 1985-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License  
along with this program. If not, see <<https://www.gnu.org/licenses/>>. \*/

/\* If ls\_mode is LS\_MULTI\_COL,  
the multi-column format is the default regardless  
of the type of output device.  
This is for the 'dir' program.

If ls\_mode is LS\_LONG\_FORMAT,  
the long format is the default regardless of the  
type of output device.  
This is for the 'vdir' program.

If ls\_mode is LS\_LS,  
the output format depends on whether the output  
device is a terminal.  
This is for the 'ls' program. \*/

/\* Written by Richard Stallman and David MacKenzie. \*/

/\* Color support by Peter Anvin <Peter.Anvin@linux.org> and Dennis  
Flaherty <dennif@denix.elk.miles.com> based on original patches by  
Greg Lee <lee@uhunix.uhcc.hawaii.edu>. \*/

```
#include <config.h>
#include <ctype.h>
#include <sys/types.h>

#include <termios.h>
#if HAVE_STROPTS_H
# include <stropts.h>
#endif
#include <sys/ioctl.h>

#ifndef WINSIZE_IN_PTEM
# include <sys/stream.h>
# include <sys/ptem.h>
#endif

#include <stdio.h>
#include <setjmp.h>
#include <pwd.h>
#include <getopt.h>
#include <signal.h>
#include <selinux/selinux.h>
#include <uchar.h>
```

```

#if HAVE_LANGINFO_CODESET
# include <langinfo.h>
#endif

/* Use SA_NOCLDSTOP as a proxy for whether the sigaction machinery is
present. */
#ifndef SA_NOCLDSTOP
#define SA_NOCLDSTOP 0
#define sigprocmask(How, Set, Oset) /* empty */
#define sigset_t int
#ifndef HAVE_SIGINTERRUPT
#define siginterrupt(sig, flag) /* empty */
#endif
#endif

/* NonStop circa 2011 lacks both SA_RESTART and siginterrupt, so don't
restart syscalls after a signal handler fires. This may cause
colors to get messed up on the screen if 'ls' is interrupted, but
that's the best we can do on such a platform. */
#ifndef SA_RESTART
#define SA_RESTART 0
#endif

#include "system.h"
#include <fnmatch.h>

#include "acl.h"
#include "argmatch.h"
#include "assure.h"
#include "c-strcasecmp.h"
#include "dev-ino.h"
#include "filenamecat.h"
#include "hard-locale.h"
#include "hash.h"
#include "human.h"
#include "filemode.h"
#include "filevercmp.h"
#include "idcache.h"
#include "ls.h"
#include "mbswidth.h"
#include "mpsort.h"
#include "obstack.h"
#include "quote.h"
#include "smack.h"
#include "stat-size.h"
#include "stat-time.h"
#include "strftime.h"
#include "xdecToInt.h"
#include "xstrtol.h"
#include "xstrtol-error.h"
#include "areadlink.h"
#include "dircolors.h"
#include "xgethostname.h"
#include "c-ctype.h"
#include "canonicalize.h"
#include "statx.h"

/* Include <sys/capability.h> last to avoid a clash of <sys/types.h>
include guards with some premature versions of libcap.

```

```

For more details, see <https://bugzilla.redhat.com/483548>. */

#ifndef HAVE_CAP
#include <sys/capability.h>
#endif

#define PROGRAM_NAME (ls_mode == LS_LS ? "ls" \
    : (ls_mode == LS_MULTI_COL \
        ? "dir" : "vdir"))

#define AUTHORS \
proper_name ("Richard M. Stallman"), \
proper_name ("David MacKenzie")

#define obstack_chunk_alloc malloc
#define obstack_chunk_free free

/* Unix-based readdir implementations have historically returned a dirent.d_ino
value that is sometimes not equal to the stat-obtained st_ino value for
that same entry. This error occurs for a readdir entry that refers
to a mount point. readdir's error is to return the inode number of
the underlying directory -- one that typically cannot be stat'ed, as
long as a file system is mounted on that directory. RELIABLE_D_INO
encapsulates whether we can use the more efficient approach of relying
on readdir-supplied d_ino values, or whether we must incur the cost of
calling stat or lstat to obtain each guaranteed-valid inode number. */

#ifndef REaddir_LIES_ABOUT_MOUNTPOINT_D_INO
#define REaddir_LIES_ABOUT_MOUNTPOINT_D_INO 1
#endif

#if REaddir_LIES_ABOUT_MOUNTPOINT_D_INO
#define RELIABLE_D_INO(dp) NOT_AN_INODE_NUMBER
#else
#define RELIABLE_D_INO(dp) D_INO (dp)
#endif

#if ! HAVE_STRUCT_STAT_ST_AUTHOR
#define st_author st_uid
#endif

enum filetype
{
    unknown,
    fifo,
    chardev,
    directory,
    blockdev,
    normal,
    symbolic_link,
    sock,
    whiteout,
    arg_directory
};

/* Display letters and indicators for each filetype.
Keep these in sync with enum filetype. */
static char const filetype_letter[] = "?pcdb-lswd";

/* Ensure that filetype and filetype_letter have the same
number of elements. */

```

```

static_assert (sizeof filetype_letter - 1 == arg_directory + 1);

#define FILETYPE_INDICATORS           \
{                                       \
    C_ORPHAN, C_FIFO, C_CHR, C_DIR, C_BLK, C_FILE, \
    C_LINK, C SOCK, C_FILE, C_DIR               \
}

enum acl_type
{
    ACL_T_NONE,
    ACL_T_LSM_CONTEXT_ONLY,
    ACL_T_YES
};

struct fileinfo
{
    /* The file name. */
    char *name;

    /* For symbolic link, name of the file linked to, otherwise zero. */
    char *linkname;

    /* For terminal hyperlinks. */
    char *absolute_name;

    struct stat stat;

    enum filetype filetype;

    /* For symbolic link and long listing, st_mode of file linked to, otherwise
     * zero. */
    mode_t linkmode;

    /* security context. */
    char *scontext;

    bool stat_ok;

    /* For symbolic link and color printing, true if linked-to file
     * exists, otherwise false. */
    bool linkok;

    /* For long listings, true if the file has an access control list,
     * or a security context. */
    enum acl_type acl_type;

    /* For color listings, true if a regular file has capability info. */
    bool has_capability;

    /* Whether file name needs quoting. tri-state with -1 == unknown. */
    int quoted;

    /* Cached screen width (including quoting). */
    size_t width;
};

#define LEN_STR_PAIR(s) sizeof (s) - 1, s

/* Null is a valid character in a color indicator (think about Epson

```

printers, for example) so we have to use a length/buffer string type. \*/

```
struct bin_str
{
    size_t len;           /* Number of bytes */
    char const *string;   /* Pointer to the same */
};

#ifndef ! HAVE_TCGETPGRP
#define tcgetpgrp(Fd) 0
#endif

static size_t quote_name (char const *name,
                         struct quoting_options const *options,
                         int needs_general_quoting,
                         const struct bin_str *color,
                         bool allow_pad, struct obstack *stack,
                         char const *absolute_name);
static size_t quote_name_buf (char **inbuf, size_t bufsize, char *name,
                           struct quoting_options const *options,
                           int needs_general_quoting, size_t *width,
                           bool *pad);
static int decode_switches (int argc, char **argv);
static bool file_ignored (char const *name);
static uintmax_t gobble_file (char const *name, enum filetype type,
                           ino_t inode, bool command_line_arg,
                           char const *dirname);
static const struct bin_str * get_color_indicator (const struct fileinfo *f,
                                                   bool symlink_target);
static bool print_color_indicator (const struct bin_str *ind);
static void put_indicator (const struct bin_str *ind);
static void add_ignore_pattern (char const *pattern);
static void attach (char *dest, char const *dirname, char const *name);
static void clear_files (void);
static void extract_dirs_from_files (char const *dirname,
                                    bool command_line_arg);
static void get_link_name (char const *filename, struct fileinfo *f,
                           bool command_line_arg);
static void indent (size_t from, size_t to);
static size_t calculate_columns (bool by_columns);
static void print_current_files (void);
static void print_dir (char const *name, char const *realname,
                      bool command_line_arg);
static size_t print_file_name_and_frills (const struct fileinfo *f,
                                         size_t start_col);
static void print_horizontal (void);
static int format_user_width (uid_t u);
static int format_group_width (gid_t g);
static void print_long_format (const struct fileinfo *f);
static void print_many_per_line (void);
static size_t print_name_with_quoting (const struct fileinfo *f,
                                      bool symlink_target,
                                      struct obstack *stack,
                                      size_t start_col);
static void prep_non_filename_text (void);
static bool print_type_indicator (bool stat_ok, mode_t mode,
                                 enum filetype type);
static void print_with_separator (char sep);
static void queue_directory (char const *name, char const *realname,
```

```

        bool command_line_arg);
static void sort_files (void);
static void parse_ls_color (void);

static int getenv_quoting_style (void);

static size_t quote_name_width (char const *name,
                               struct quoting_options const *options,
                               int needs_general_quoting);

/* Initial size of hash table.
Most hierarchies are likely to be shallower than this. */
enum { INITIAL_TABLE_SIZE = 30 };

/* The set of 'active' directories, from the current command-line argument
to the level in the hierarchy at which files are being listed.
A directory is represented by its device and inode numbers (struct dev_ino).
A directory is added to this set when ls begins listing it or its
entries, and it is removed from the set just after ls has finished
processing it. This set is used solely to detect loops, e.g., with
mkdir loop; cd loop; ln -s ..//loop sub; ls -RL */
static Hash_table *active_dir_set;

#define LOOP_DETECT (!!active_dir_set)

/* The table of files in the current directory:

'cwd_file' points to a vector of 'struct fileinfo', one per file.
'cwd_n_alloc' is the number of elements space has been allocated for.
'cwd_n_used' is the number actually in use. */

/* Address of block containing the files that are described. */
static struct fileinfo * cwd_file;

/* Length of block that 'cwd_file' points to, measured in files. */
static size_t cwd_n_alloc;

/* Index of first unused slot in 'cwd_file'. */
static size_t cwd_n_used;

/* Whether files needs may need padding due to quoting. */
static bool cwd_some_quoted;

/* Whether quoting style _may_ add outer quotes,
and whether aligning those is useful. */
static bool align_variable_outer_quotes;

/* Vector of pointers to files, in proper sorted order, and the number
of entries allocated for it. */
static void **sorted_file;
static size_t sorted_file_alloc;

/* When true, in a color listing, color each symlink name according to the
type of file it points to. Otherwise, color them according to the 'In'
directive in LS_COLORS. Dangling (orphan) symlinks are treated specially,
regardless. This is set when 'In=target' appears in LS_COLORS. */

static bool color_symlink_as_referent;

static char const *hostname;

```

```

/* Mode of appropriate file for coloring. */
static mode_t
file_or_link_mode (struct fileinfo const *file)
{
    return (color_symlink_as_referent && file->linkok
            ? file->linkmode : file->stat.st_mode);
}

/* Record of one pending directory waiting to be listed. */

struct pending
{
    char *name;
    /* If the directory is actually the file pointed to by a symbolic link we
       were told to list, 'realname' will contain the name of the symbolic
       link, otherwise zero. */
    char *realname;
    bool command_line_arg;
    struct pending *next;
};

static struct pending *pending_dirs;

/* Current time in seconds and nanoseconds since 1970, updated as
   needed when deciding whether a file is recent. */

static struct timespec current_time;

static bool print_scontext;
static char UNKNOWN_SECURITY_CONTEXT[] = "?";

/* Whether any of the files has an ACL. This affects the width of the
   mode column. */

static bool any_has_acl;

/* The number of columns to use for columns containing inode numbers,
   block sizes, link counts, owners, groups, authors, major device
   numbers, minor device numbers, and file sizes, respectively. */

static int inode_number_width;
static int block_size_width;
static int nlink_width;
static int scontext_width;
static int owner_width;
static int group_width;
static int author_width;
static int major_device_number_width;
static int minor_device_number_width;
static int file_size_width;

/* Option flags */

/* long_format for lots of info, one per line.
   one_per_line for just names, one per line.
   many_per_line for just names, many per line, sorted vertically.
   horizontal for just names, many per line, sorted horizontally.
   with_commas for just names, many per line, separated by commas.

```

```
-l (and other options that imply -l), -1, -C, -x and -m control  
this parameter. */
```

```
enum format
{
    long_format,           /* -l and other options that imply -l */
    one_per_line,          /* -1 */
    many_per_line,         /* -C */
    horizontal,            /* -x */
    with_commas           /* -m */
};

static enum format format;

/* 'full-iso' uses full ISO-style dates and times. 'long-iso' uses longer
ISO-style timestamps, though shorter than 'full-iso'. 'iso' uses shorter
ISO-style timestamps. 'locale' uses locale-dependent timestamps. */
enum time_style
{
    full_iso_time_style, /* --time-style=full-iso */
    long_iso_time_style, /* --time-style=long-iso */
    iso_time_style,      /* --time-style=iso */
    locale_time_style    /* --time-style=locale */
};

static char const *const time_style_args[] =
{
    "full-iso", "long-iso", "iso", "locale", nullptr
};
static enum time_style const time_style_types[] =
{
    full_iso_time_style, long_iso_time_style, iso_time_style,
    locale_time_style
};
ARGMATCH_VERIFY (time_style_args, time_style_types);

/* Type of time to print or sort by. Controlled by -c and -u.
The values of each item of this enum are important since they are
used as indices in the sort functions array (see sort_files()). */

enum time_type
{
    time_mtime = 0,        /* default */
    time_ctime,            /* -c */
    time_atime,            /* -u */
    time_btime,             /* birth time */
    time_numtypes          /* the number of elements of this enum */
};

static enum time_type time_type;

/* The file characteristic to sort by. Controlled by -t, -S, -U, -X, -v.
The values of each item of this enum are important since they are
used as indices in the sort functions array (see sort_files()). */

enum sort_type
{
    sort_name = 0,          /* default */
    sort_extension,          /* -X */
};
```

```

sort_width,
sort_size,           /* -S */
sort_version,        /* -v */
sort_time,           /* -t; must be second to last */
sort_none,           /* -U; must be last */
sort_numtypes        /* the number of elements of this enum */
};

static enum sort_type sort_type;

/* Direction of sort.
false means highest first if numeric,
lowest first if alphabetic;
these are the defaults.
true means the opposite order in each case. -r */

static bool sort_reverse;

/* True means to display owner information. -g turns this off. */

static bool print_owner = true;

/* True means to display author information. */

static bool print_author;

/* True means to display group information. -G and -o turn this off. */

static bool print_group = true;

/* True means print the user and group id's as numbers rather
than as names. -n */

static bool numeric_ids;

/* True means mention the size in blocks of each file. -s */

static bool print_block_size;

/* Human-readable options for output, when printing block counts. */
static int human_output_opts;

/* The units to use when printing block counts. */
static uintmax_t output_block_size;

/* Likewise, but for file sizes. */
static int file_human_output_opts;
static uintmax_t file_output_block_size = 1;

/* Follow the output with a special string. Using this format,
Emacs' dired mode starts up twice as fast, and can handle all
strange characters in file names. */
static bool dired;

/* 'none' means don't mention the type of files.
'slash' means mention directories only, with a '/'.
'file_type' means mention file types.
'classify' means mention file types and mark executables.

Controlled by -F, -p, and --indicator-style. */

```

```

enum indicator_style
{
    none = 0, /* --indicator-style=none (default) */
    slash,     /* -p, --indicator-style=slash */
    file_type, /* --indicator-style=file-type */
    classify   /* -F, --indicator-style=classify */
};

static enum indicator_style indicator_style;

/* Names of indicator styles. */
static char const *const indicator_style_args[] =
{
    "none", "slash", "file-type", "classify", nullptr
};
static enum indicator_style const indicator_style_types[] =
{
    none, slash, file_type, classify
};
ARGMATCH_VERIFY(indicator_style_args, indicator_style_types);

/* True means use colors to mark types. Also define the different
colors as well as the stuff for the LS_COLORS environment variable.
The LS_COLORS variable is now in a termcap-like format. */

static bool print_with_color;

static bool print_hyperlink;

/* Whether we used any colors in the output so far. If so, we will
need to restore the default color later. If not, we will need to
call prep_non_filename_text before using color for the first time. */

static bool used_color = false;

enum when_type
{
    when_never,           /* 0: default or --color=never */
    when_always,          /* 1: --color=always */
    when_if_tty           /* 2: --color=tty */
};

enum Dereference_symlink
{
    DEREF_UNDEFINED = 0,      /* default */
    DEREF_NEVER,
    DEREF_COMMAND_LINE_ARGUMENTS, /* -H */
    DEREF_COMMAND_LINE_SYMLINK_TO_DIR, /* the default, in certain cases */
    DEREF_ALWAYS           /* -L */
};

enum indicator_no
{
    C_LEFT, C_RIGHT, C_END, C_RESET, C_NORM, C_FILE, C_DIR, C_LINK,
    C_FIFO, C_SOCK,
    C_BLK, C_CHR, C_MISSING, C_ORPHAN, C_EXEC, C_DOOR, C_SETUID, C_SETGID,
    C_STICKY, C_OTHER_WRITABLE, C_STICKY_OTHER_WRITABLE, C_CAP,
    C_MULTIHARDLINK,
    C_CLR_TO_EOL
}

```

```

};

static char const *const indicator_name[]=
{
    "lc", "rc", "ec", "rs", "no", "fi", "di", "ln", "pi", "so",
    "bd", "cd", "mi", "or", "ex", "do", "su", "sg", "st",
    "ow", "tw", "ca", "mh", "cl", nullptr
};

struct color_ext_type
{
    struct bin_str ext;           /* The extension we're looking for */
    struct bin_str seq;          /* The sequence to output when we do */
    bool exact_match;            /* Whether to compare case insensitively */
    struct color_ext_type *next; /* Next in list */
};

static struct bin_str color_indicator[] =
{
    { LEN_STR_PAIR ("\033[") },           /* lc: Left of color sequence */
    { LEN_STR_PAIR ("m") },                /* rc: Right of color sequence */
    { 0, nullptr },                      /* ec: End color (replaces lc+rs+rc) */
    { LEN_STR_PAIR ("0") },                /* rs: Reset to ordinary colors */
    { 0, nullptr },                      /* no: Normal */
    { 0, nullptr },                      /* fi: File: default */
    { LEN_STR_PAIR ("01;34") },           /* di: Directory: bright blue */
    { LEN_STR_PAIR ("01;36") },           /* ln: Symlink: bright cyan */
    { LEN_STR_PAIR ("33") },              /* pi: Pipe: yellow/brown */
    { LEN_STR_PAIR ("01;35") },           /* so: Socket: bright magenta */
    { LEN_STR_PAIR ("01;33") },           /* bd: Block device: bright yellow */
    { LEN_STR_PAIR ("01;33") },           /* cd: Char device: bright yellow */
    { 0, nullptr },                      /* mi: Missing file: undefined */
    { 0, nullptr },                      /* or: Orphaned symlink: undefined */
    { LEN_STR_PAIR ("01;32") },           /* ex: Executable: bright green */
    { LEN_STR_PAIR ("01;35") },           /* do: Door: bright magenta */
    { LEN_STR_PAIR ("37;41") },           /* su: setuid: white on red */
    { LEN_STR_PAIR ("30;43") },           /* sg: setgid: black on yellow */
    { LEN_STR_PAIR ("37;44") },           /* st: sticky: black on blue */
    { LEN_STR_PAIR ("34;42") },           /* ow: other-writable: blue on green */
    { LEN_STR_PAIR ("30;42") },           /* tw: ow w/ sticky: black on green */
    { 0, nullptr },                      /* ca: disabled by default */
    { 0, nullptr },                      /* mh: disabled by default */
    { LEN_STR_PAIR ("\033[K") },          /* cl: clear to end of line */
};

/* A list mapping file extensions to corresponding display sequence. */
static struct color_ext_type *color_ext_list = nullptr;

/* Buffer for color sequences */
static char *color_buf;

/* True means to check for orphaned symbolic link, for displaying
colors, or to group symlink to directories with other dirs. */

static bool check_symlink_mode;

/* True means mention the inode number of each file. -i */

static bool print_inode;

```

```

/* What to do with symbolic links. Affected by -d, -F, -H, -l (and
other options that imply -l), and -L. */

static enum Dereference_symlink dereference;

/* True means when a directory is found, display info on its
contents. -R */

static bool recursive;

/* True means when an argument is a directory name, display info
on it itself. -d */

static bool immediate_dirs;

/* True means that directories are grouped before files. */

static bool directories_first;

/* Which files to ignore. */

static enum
{
    /* Ignore files whose names start with '.', and files specified by
       --hide and --ignore. */
    IGNORE_DEFAULT = 0,
    /* Ignore '.', '..', and files specified by --ignore. */
    IGNORE_DOT_AND_DOTDOT,
    /* Ignore only files specified by --ignore. */
    IGNORE_MINIMAL
} ignore_mode;

/* A linked list of shell-style globbing patterns. If a non-argument
file name matches any of these patterns, it is ignored.
Controlled by -I. Multiple -I options accumulate.
The -B option adds '*~' and '.*~' to this list. */

struct ignore_pattern
{
    char const *pattern;
    struct ignore_pattern *next;
};

static struct ignore_pattern *ignore_patterns;

/* Similar to IGNORE_PATTERNS, except that -a or -A causes this
variable itself to be ignored. */
static struct ignore_pattern *hide_patterns;

/* True means output nongraphic chars in file names as '?'.
(-q, --hide-control-chars)
qmark_funny_chars and the quoting style (-Q, --quoting-style=WORD) are
independent. The algorithm is: first, obey the quoting style to get a
string representing the file name; then, if qmark_funny_chars is set,
replace all nonprintable chars in that string with '?'. It's necessary
to replace nonprintable chars even in quoted strings, because we don't
want to mess up the terminal if control chars get sent to it, and some
quoting methods pass through control chars as-is. */

```

```

static bool qmark_funny_chars;

/* Quoting options for file and dir name output. */

static struct quoting_options *filename_quoting_options;
static struct quoting_options *dirname_quoting_options;

/* The number of chars per hardware tab stop. Setting this to zero
inhibits the use of TAB characters for separating columns. -T */
static size_t tabsize;

/* True means print each directory name before listing it. */

static bool print_dir_name;

/* The line length to use for breaking lines in many-per-line format.
Can be set with -w. If zero, there is no limit. */

static size_t line_length;

/* The local time zone rules, as per the TZ environment variable. */

static timezone_t localtz;

/* If true, the file listing format requires that stat be called on
each file. */

static bool format_needs_stat;

/* Similar to 'format_needs_stat', but set if only the file type is
needed. */

static bool format_needs_type;

/* An arbitrary limit on the number of bytes in a printed timestamp.
This is set to a relatively small value to avoid the need to worry
about denial-of-service attacks on servers that run "ls" on behalf
of remote clients. 1000 bytes should be enough for any practical
timestamp format. */

enum { TIME_STAMP_LEN_MAXIMUM = MAX (1000, INT_STRLEN_BOUND (time_t)) };

/* strftime formats for non-recent and recent files, respectively, in
-l output. */

static char const *long_time_format[2] =
{
    /* strftime format for non-recent files (older than 6 months), in
    -l output. This should contain the year, month and day (at
    least), in an order that is understood by people in your
    locale's territory. Please try to keep the number of used
    screen columns small, because many people work in windows with
    only 80 columns. But make this as wide as the other string
    below, for recent files. */
    /* TRANSLATORS: ls output needs to be aligned for ease of reading,
    so be wary of using variable width fields from the locale.
    Note %b is handled specially by ls and aligned correctly.
    Note also that specifying a width as in %5b is erroneous as strftime
    will count bytes rather than characters in multibyte locales. */
    N_(""%b %e %Y"),

```

```

/* strftime format for recent files (younger than 6 months), in -l
output. This should contain the month, day and time (at
least), in an order that is understood by people in your
locale's territory. Please try to keep the number of used
screen columns small, because many people work in windows with
only 80 columns. But make this as wide as the other string
above, for non-recent files. */
/* TRANSLATORS: ls output needs to be aligned for ease of reading,
so be wary of using variable width fields from the locale.
Note %b is handled specially by ls and aligned correctly.
Note also that specifying a width as in %5b is erroneous as strftime
will count bytes rather than characters in multibyte locales. */
N_(""%b %e %H:%M")
};

/* The set of signals that are caught. */

static sigset_t caught_signals;

/* If nonzero, the value of the pending fatal signal. */

static sig_atomic_t volatile interrupt_signal;

/* A count of the number of pending stop signals that have been received. */

static sig_atomic_t volatile stop_signal_count;

/* Desired exit status. */

static int exit_status;

/* Exit statuses. */
enum
{
    /* "ls" had a minor problem. E.g., while processing a directory,
ls obtained the name of an entry via readdir, yet was later
unable to stat that name. This happens when listing a directory
in which entries are actively being removed or renamed. */
    LS_MINOR_PROBLEM = 1,
    /* "ls" had more serious trouble (e.g., memory exhausted, invalid
option or failure to stat a command line argument. */
    LS_FAILURE = 2
};

/* For long options that have no equivalent short option, use a
non-character as a pseudo short option, starting with CHAR_MAX + 1. */
enum
{
    AUTHOR_OPTION = CHAR_MAX + 1,
    BLOCK_SIZE_OPTION,
    COLOR_OPTION,
    DEREference_COMMAND_LINE_SYMLINK_TO_DIR_OPTION,
    FILE_TYPE_INDICATOR_OPTION,
    FORMAT_OPTION,
    FULL_TIME_OPTION,
    GROUP_DIRECTORIES_FIRST_OPTION,
    HIDE_OPTION,
    HYPERLINK_OPTION,
    INDICATOR_STYLE_OPTION,

```

```

QUOTING_STYLE_OPTION,
SHOW_CONTROL_CHARS_OPTION,
SI_OPTION,
SORT_OPTION,
TIME_OPTION,
TIME_STYLE_OPTION,
ZERO_OPTION,
};

static struct option const long_options[] =
{
{"all", no_argument, nullptr, 'a'},
 {"escape", no_argument, nullptr, 'b'},
 {"directory", no_argument, nullptr, 'd'},
 {"dired", no_argument, nullptr, 'D'},
 {"full-time", no_argument, nullptr, FULL_TIME_OPTION},
 {"group-directories-first", no_argument, nullptr,
 GROUP_DIRECTORIES_FIRST_OPTION},
 {"human-readable", no_argument, nullptr, 'h'},
 {"inode", no_argument, nullptr, 'i'},
 {"kibibytes", no_argument, nullptr, 'k'},
 {"numeric-uid-gid", no_argument, nullptr, 'n'},
 {"no-group", no_argument, nullptr, 'G'},
 {"hide-control-chars", no_argument, nullptr, 'q'},
 {"reverse", no_argument, nullptr, 'r'},
 {"size", no_argument, nullptr, 's'},
 {"width", required_argument, nullptr, 'w'},
 {"almost-all", no_argument, nullptr, 'A'},
 {"ignore-backups", no_argument, nullptr, 'B'},
 {"classify", optional_argument, nullptr, 'F'},
 {"file-type", no_argument, nullptr, FILE_TYPE_INDICATOR_OPTION},
 {"si", no_argument, nullptr, SI_OPTION},
 {"dereference-command-line", no_argument, nullptr, 'H'},
 {"dereference-command-line-symlink-to-dir", no_argument, nullptr,
 DEREference_COMMAND_LINE_SYMLINK_TO_DIR_OPTION},
 {"hide", required_argument, nullptr, HIDE_OPTION},
 {"ignore", required_argument, nullptr, 'I'},
 {"indicator-style", required_argument, nullptr, INDICATOR_STYLE_OPTION},
 {"dereference", no_argument, nullptr, 'L'},
 {"literal", no_argument, nullptr, 'N'},
 {"quote-name", no_argument, nullptr, 'Q'},
 {"quoting-style", required_argument, nullptr, QUOTING_STYLE_OPTION},
 {"recursive", no_argument, nullptr, 'R'},
 {"format", required_argument, nullptr, FORMAT_OPTION},
 {"show-control-chars", no_argument, nullptr, SHOW_CONTROL_CHARS_OPTION},
 {"sort", required_argument, nullptr, SORT_OPTION},
 {"tabsize", required_argument, nullptr, 'T'},
 {"time", required_argument, nullptr, TIME_OPTION},
 {"time-style", required_argument, nullptr, TIME_STYLE_OPTION},
 {"zero", no_argument, nullptr, ZERO_OPTION},
 {"color", optional_argument, nullptr, COLOR_OPTION},
 {"hyperlink", optional_argument, nullptr, HYPERLINK_OPTION},
 {"block-size", required_argument, nullptr, BLOCK_SIZE_OPTION},
 {"context", no_argument, 0, 'Z'},
 {"author", no_argument, nullptr, AUTHOR_OPTION},
 {GETOPT_HELP_OPTION_DECL},
 {GETOPT_VERSION_OPTION_DECL},
 {nullptr, 0, nullptr, 0}
};

```

```

static char const *const format_args[] =
{
    "verbose", "long", "commas", "horizontal", "across",
    "vertical", "single-column", nullptr
};
static enum format const format_types[] =
{
    long_format, long_format, with_commas, horizontal, horizontal,
    many_per_line, one_per_line
};
ARGMATCH_VERIFY (format_args, format_types);

static char const *const sort_args[] =
{
    "none", "time", "size", "extension", "version", "width", nullptr
};
static enum sort_type const sort_types[] =
{
    sort_none, sort_time, sort_size, sort_extension, sort_version, sort_width
};
ARGMATCH_VERIFY (sort_args, sort_types);

static char const *const time_args[] =
{
    "atime", "access", "use",
    "ctime", "status",
    "mtime", "modification",
    "birth", "creation",
    nullptr
};
static enum time_type const time_types[] =
{
    time_atime, time_atime, time_atime,
    time_ctime, time_ctime,
    time_mtime, time_mtime,
    time_btime, time_btime,
};
ARGMATCH_VERIFY (time_args, time_types);

static char const *const when_args[] =
{
    /* force and none are for compatibility with another color-ls version */
    "always", "yes", "force",
    "never", "no", "none",
    "auto", "tty", "if-tty", nullptr
};
static enum when_type const when_types[] =
{
    when_always, when_always, when_always,
    when_never, when_never, when_never,
    when_if_tty, when_if_tty, when_if_tty
};
ARGMATCH_VERIFY (when_args, when_types);

/* Information about filling a column. */
struct column_info
{
    bool valid_len;
    size_t line_len;
    size_t *col_arr;
}

```

```

};

/* Array with information about column fullness. */
static struct column_info *column_info;

/* Maximum number of columns ever possible for this display. */
static size_t max_idx;

/* The minimum width of a column is 3: 1 character for the name and 2
for the separating white space. */
enum { MIN_COLUMN_WIDTH = 3 };

/* This zero-based index is for the --dired option. It is incremented
for each byte of output generated by this program so that the beginning
and ending indices (in that output) of every file name can be recorded
and later output themselves. */
static off_t dired_pos;

static void
dired_outbyte (char c)
{
dired_pos++;
putchar (c);
}

/* Output the buffer S, of length S_LEN, and increment DIRED_POS by S_LEN. */
static void
dired_outbuf (char const *s, size_t s_len)
{
dired_pos += s_len;
fwrite (s, sizeof *s, s_len, stdout);
}

/* Output the string S, and increment DIRED_POS by its length. */
static void
dired_outstring (char const *s)
{
dired_outbuf (s, strlen (s));
}

static void
dired_indent (void)
{
if (dired)
    dired_outstring ("  ");
}

/* With --dired, store pairs of beginning and ending indices of file names. */
static struct obstack dired_obstack;

/* With --dired, store pairs of beginning and ending indices of any
directory names that appear as headers (just before 'total' line)
for lists of directory entries. Such directory names are seen when
listing hierarchies using -R and when a directory is listed with at
least one other command line argument. */
static struct obstack subdired_obstack;

/* Save the current index on the specified obstack, OBS. */
static void

```

```

push_current_dired_pos (struct obstack *obs)
{
if (dired)
    obstack_grow (obs, &dired_pos, sizeof dired_pos);
}

/* With -R, this stack is used to help detect directory cycles.
The device/inode pairs on this stack mirror the pairs in the
active_dir_set hash table. */
static struct obstack dev_ino_obstack;

/* Push a pair onto the device/inode stack. */
static void
dev_ino_push (dev_t dev, ino_t ino)
{
void *vdi;
struct dev_ino *di;
int dev_ino_size = sizeof *di;
obstack_blank (&dev_ino_obstack, dev_ino_size);
vdi = obstack_next_free (&dev_ino_obstack);
di = vdi;
di--;
di->st_dev = dev;
di->st_ino = ino;
}

/* Pop a dev/ino struct off the global dev_ino_obstack
and return that struct. */
static struct dev_ino
dev_ino_pop (void)
{
void *vdi;
struct dev_ino *di;
int dev_ino_size = sizeof *di;
affirm (dev_ino_size <= obstack_object_size (&dev_ino_obstack));
obstack_blank_fast (&dev_ino_obstack, -dev_ino_size);
vdi = obstack_next_free (&dev_ino_obstack);
di = vdi;
return *di;
}

static void
assert_matching_dev_ino (char const *name, struct dev_ino di)
{
MAYBE_UNUSED struct stat sb;
assure (0 <= stat (name, &sb));
assure (sb.st_dev == di.st_dev);
assure (sb.st_ino == di.st_ino);
}

static char eolbyte = '\n';

/* Write to standard output PREFIX, followed by the quoting style and
a space-separated list of the integers stored in OS all on one line. */

static void
dired_dump_obstack (char const *prefix, struct obstack *os)
{
size_t n_pos;

```

```

n_pos = obstack_object_size (os) / sizeof (dired_pos);
if (n_pos > 0)
{
    off_t *pos = obstack_finish (os);
    fputs (prefix, stdout);
    for (size_t i = 0; i < n_pos; i++)
    {
        intmax_t p = pos[i];
        printf (" %jd", p);
    }
    putchar ('\n');
}
}

/* Return the platform birthtime member of the stat structure,
or fallback to the mtime member, which we have populated
from the statx structure or reset to an invalid timestamp
where birth time is not supported. */
static struct timespec
get_stat_btime (struct stat const *st)
{
    struct timespec btimespec;

#ifndef HAVE_STATX
btimespec = get_stat_mtime (st);
#else
btimespec = get_stat_birthtime (st);
#endif

    return btimespec;
}

#ifndef HAVE_STATX
ATTRIBUTE_PURE
static unsigned int
time_type_to_statx (void)
{
    switch (time_type)
    {
        case time_ctime:
            return STATX_CTIME;
        case time_mtime:
            return STATX_MTIME;
        case time_atime:
            return STATX_ATIME;
        case time_btime:
            return STATX_BTIME;
        default:
            unreachable ();
    }
    return 0;
}

ATTRIBUTE_PURE
static unsigned int
calc_req_mask (void)
{
    unsigned int mask = STATX_MODE;

    if (print_inode)

```

```

mask |= STATX_INO;

if (print_block_size)
    mask |= STATX_BLOCKS;

if (format == long_format) {
    mask |= STATX_NLINK | STATX_SIZE | time_type_to_statx ();
    if (print_owner || print_author)
        mask |= STATX_UID;
    if (print_group)
        mask |= STATX_GID;
}

switch (sort_type)
{
case sort_none:
case sort_name:
case sort_version:
case sort_extension:
case sort_width:
break;
case sort_time:
    mask |= time_type_to_statx ();
    break;
case sort_size:
    mask |= STATX_SIZE;
    break;
default:
    unreachable ();
}

return mask;
}

static int
do_statx (int fd, char const *name, struct stat *st, int flags,
          unsigned int mask)
{
struct statx stx;
bool want_btime = mask & STATX_BTIME;
int ret = statx (fd, name, flags | AT_NO_AUTOMOUNT, mask, &stx);
if (ret >= 0)
{
    statx_to_stat (&stx, st);
    /* Since we only need one timestamp type,
       store birth time in st_mtim. */
    if (want_btime)
    {
        if (stx.stx_mask & STATX_BTIME)
            st->st_mtim = statx_timestamp_to_timespec (stx.stx_btime);
        else
            st->st_mtim.tv_sec = st->st_mtim.tv_nsec = -1;
    }
}

return ret;
}

static int
do_stat (char const *name, struct stat *st)

```

```

{
return do_statx(AT_FDCWD, name, st, 0, calc_req_mask ());
}

static int
do_lstat (char const *name, struct stat *st)
{
return do_statx (AT_FDCWD, name, st, AT_SYMLINK_NOFOLLOW, calc_req_mask ());
}

static int
stat_for_mode (char const *name, struct stat *st)
{
return do_statx (AT_FDCWD, name, st, 0, STATX_MODE);
}

/* dev+ino should be static, so no need to sync with backing store */
static int
stat_for_ino (char const *name, struct stat *st)
{
return do_statx (AT_FDCWD, name, st, 0, STATX_INO);
}

static int
fstat_for_ino (int fd, struct stat *st)
{
return do_statx (fd, "", st, AT_EMPTY_PATH, STATX_INO);
}
#else
static int
do_stat (char const *name, struct stat *st)
{
return stat (name, st);
}

static int
do_lstat (char const *name, struct stat *st)
{
return lstat (name, st);
}

static int
stat_for_mode (char const *name, struct stat *st)
{
return stat (name, st);
}

static int
stat_for_ino (char const *name, struct stat *st)
{
return stat (name, st);
}

static int
fstat_for_ino (int fd, struct stat *st)
{
return fstat (fd, st);
}
#endif

```

```

/* Return the address of the first plain %b spec in FMT, or nullptr if
there is no such spec. %5b etc. do not match, so that user
widths/flags are honored. */

ATTRIBUTE PURE
static char const *
first_percent_b (char const *fmt)
{
for (; *fmt; fmt++)
    if (fmt[0] == '%')
        switch (fmt[1])
        {
        case 'b': return fmt;
        case '%': fmt++; break;
        }
return nullptr;
}

static char RFC3986[256];
static void
file_escape_init (void)
{
for (int i = 0; i < 256; i++)
    RFC3986[i] |= c_isalnum (i) || i == '~' || i == '-' || i == '.' || i == '_';
}

enum { MBSWIDTH_FLAGS = MBSW_REJECT_INVALID |
MBSW_REJECT_UNPRINTABLE };

/* Read the abbreviated month names from the locale, to align them
and to determine the max width of the field and to truncate names
greater than our max allowed.
Note even though this handles multibyte locales correctly
it's not restricted to them as single byte locales can have
variable width abbreviated months and also precomputing/caching
the names was seen to increase the performance of ls significantly. */

/* abformat[RECENT][MON] is the format to use for timestamps with
recentness RECENT and month MON. */
enum { ABFORMAT_SIZE = 128 };
static char abformat[2][12][ABFORMAT_SIZE];
/* True if precomputed formats should be used. This can be false if
nl_langinfo fails, if a format or month abbreviation is unusually
long, or if a month abbreviation contains '%'. */
static bool use_abformat;

/* Store into ABMON the abbreviated month names, suitably aligned.
Return true if successful. */

static bool
abmon_init (char abmon[12][ABFORMAT_SIZE])
{
#ifndef HAVE_NL_LANGINFO
return false;
#else
int max_mon_width = 0;
int mon_width[12];
int mon_len[12];

for (int i = 0; i < 12; i++)

```

```

{
char const *abbr = nl_langinfo (ABMON_1 + i);
mon_len[i] = strnlen (abbr, ABFORMAT_SIZE);
if (mon_len[i] == ABFORMAT_SIZE)
    return false;
if (strchr (abbr, '%'))
    return false;
mon_width[i] = mbswidth (strcpy (abmon[i], abbr), MBSWIDTH_FLAGS);
if (mon_width[i] < 0)
    return false;
max_mon_width = MAX (max_mon_width, mon_width[i]);
}

for (int i = 0; i < 12; i++)
{
int fill = max_mon_width - mon_width[i];
if (ABFORMAT_SIZE - mon_len[i] <= fill)
    return false;
bool align_left = !isdigit (to_uchar (abmon[i][0]));
int fill_offset;
if (align_left)
    fill_offset = mon_len[i];
else
{
    memmove (abmon[i] + fill, abmon[i], mon_len[i]);
    fill_offset = 0;
}
memset (abmon[i] + fill_offset, ' ', fill);
abmon[i][mon_len[i] + fill] = '\0';
}

return true;
#endif
}

/* Initialize ABFORMAT and USE_ABFORMAT. */

static void
abformat_init (void)
{
char const *pb[2];
for (int recent = 0; recent < 2; recent++)
    pb[recent] = first_percent_b (long_time_format[recent]);
if (! (pb[0] || pb[1]))
    return;

char abmon[12][ABFORMAT_SIZE];
if (! abmon_init (abmon))
    return;

for (int recent = 0; recent < 2; recent++)
{
char const *fmt = long_time_format[recent];
for (int i = 0; i < 12; i++)
{
    char *nfmt = abformat[recent][i];
    int nbytes;

    if (! pb[recent])
        nbytes = snprintf (nfmt, ABFORMAT_SIZE, "%s", fmt);
}
}
}

```

```

    else
    {
        if (! (pb[recent] - fmt <= MIN (ABFORMAT_SIZE, INT_MAX)))
            return;
        int prefix_len = pb[recent] - fmt;
        nbytes = snprintf (nfmt, ABFORMAT_SIZE, "%.*s%s%s",
                           prefix_len, fmt, abmon[i], pb[recent] + 2);
    }

    if (! (0 <= nbytes && nbytes < ABFORMAT_SIZE))
        return;
}

use_abformat = true;
}

static size_t
dev_ino_hash (void const *x, size_t table_size)
{
struct dev_ino const *p = x;
return (uintmax_t) p->st_ino % table_size;
}

static bool
dev_ino_compare (void const *x, void const *y)
{
struct dev_ino const *a = x;
struct dev_ino const *b = y;
return PSAME_INODE (a, b);
}

static void
dev_ino_free (void *x)
{
free (x);
}

/* Add the device/inode pair (P->st_dev/P->st_ino) to the set of
active directories. Return true if there is already a matching
entry in the table. */

static bool
visit_dir (dev_t dev, ino_t ino)
{
struct dev_ino *ent;
struct dev_ino *ent_from_table;
bool found_match;

ent = xmalloc (sizeof *ent);
ent->st_ino = ino;
ent->st_dev = dev;

/* Attempt to insert this entry into the table. */
ent_from_table = hash_insert (active_dir_set, ent);

if (ent_from_table == nullptr)
{
    /* Insertion failed due to lack of memory. */
    xalloc_die ();
}

```

```

    }

found_match = (ent_from_table != ent);

if (found_match)
{
    /* ent was not inserted, so free it. */
    free (ent);
}

return found_match;
}

static void
free_pending_ent (struct pending *p)
{
free (p->name);
free (p->realname);
free (p);
}

static bool
is_colored (enum indicator_no type)
{
size_t len = color_indicator[type].len;
char const *s = color_indicator[type].string;
return ! (len == 0
    || (len == 1 && STRNCMP_LIT (s, "0") == 0)
    || (len == 2 && STRNCMP_LIT (s, "00") == 0));
}

static void
restore_default_color (void)
{
put_indicator (&color_indicator[C_LEFT]);
put_indicator (&color_indicator[C_RIGHT]);
}

static void
set_normal_color (void)
{
if (print_with_color && is_colored (C_NORM))
{
    put_indicator (&color_indicator[C_LEFT]);
    put_indicator (&color_indicator[C_NORM]);
    put_indicator (&color_indicator[C_RIGHT]);
}
}

/* An ordinary signal was received; arrange for the program to exit. */

static void
sighandler (int sig)
{
if (! SA_NOCLDSTOP)
    signal (sig, SIG_IGN);
if (! interrupt_signal)
    interrupt_signal = sig;
}

```

```

/* A SIGTSTP was received; arrange for the program to suspend itself. */

static void
stophandler (int sig)
{
if (! SA_NOCLDSTOP)
    signal (sig, stophandler);
if (! interrupt_signal)
    stop_signal_count++;
}

/* Process any pending signals. If signals are caught, this function
should be called periodically. Ideally there should never be an
unbounded amount of time when signals are not being processed.
Signal handling can restore the default colors, so callers must
immediately change colors after invoking this function. */

static void
process_signals (void)
{
while (interrupt_signal || stop_signal_count)
{
    int sig;
    int stops;
    sigset_t oldset;

    if (used_color)
        restore_default_color ();
    fflush (stdout);

    sigprocmask (SIG_BLOCK, &caught_signals, &oldset);

    /* Reload interrupt_signal and stop_signal_count, in case a new
       signal was handled before sigprocmask took effect. */
    sig = interrupt_signal;
    stops = stop_signal_count;

    /* SIGTSTP is special, since the application can receive that signal
       more than once. In this case, don't set the signal handler to the
       default. Instead, just raise the uncatchable SIGSTOP. */
    if (stops)
    {
        stop_signal_count = stops - 1;
        sig = SIGSTOP;
    }
    else
        signal (sig, SIG_DFL);

    /* Exit or suspend the program. */
    raise (sig);
    sigprocmask (SIG_SETMASK, &oldset, nullptr);

    /* If execution reaches here, then the program has been
       continued (after being suspended). */
}
}

/* Setup signal handlers if INIT is true,
otherwise restore to the default. */

```

```

static void
signal_setup (bool init)
{
/* The signals that are trapped, and the number of such signals. */
static int const sig[] =
{
    /* This one is handled specially. */
    SIGTSTP,

    /* The usual suspects. */
    SIGNALRM, SIGHUP, SIGINT, SIGPIPE, SIGQUIT, SIGTERM,
#endifif SIGPOLL
    SIGPOLL,
#endifif
#ifndefif SIGPROF
    SIGPROF,
#endifif
#ifndefif SIGVTALRM
    SIGVTALRM,
#endifif
#ifndefif SIGXCPU
    SIGXCPU,
#endifif
#ifndefif SIGXFSZ
    SIGXFSZ,
#endifif
};

enum { nsigs = ARRAY_CARDINALITY (sig) };

#ifndefif ! SA_NOCLDSTOP
static bool caught_sig[nsigs];
#endifif

int j;

if (init)
{
#ifndefif SA_NOCLDSTOP
    struct sigaction act;

    sigemptyset (&caught_signals);
    for (j = 0; j < nsigs; j++)
    {
        sigaction (sig[j], nullptr, &act);
        if (act.sa_handler != SIG_IGN)
            sigaddset (&caught_signals, sig[j]);
    }

    act.sa_mask = caught_signals;
    act.sa_flags = SA_RESTART;

    for (j = 0; j < nsigs; j++)
        if (sigismember (&caught_signals, sig[j]))
        {
            act.sa_handler = sig[j] == SIGTSTP ? stophandler : sighandler;
            sigaction (sig[j], &act, nullptr);
        }
#else
    for (j = 0; j < nsigs; j++)
    {

```

```

caught_sig[j] = (signal (sig[j], SIG_IGN) != SIG_IGN);
if (caught_sig[j])
{
    signal (sig[j], sig[j] == SIGTSTP ? stophandler : sighandler);
    siginterrupt (sig[j], 0);
}
}
#endif
}
else /* restore. */
{
#if SA_NOCLDSTOP
    for (j = 0; j < nsigs; j++)
        if (sigismember (&caught_signals, sig[j]))
            signal (sig[j], SIG_DFL);
#else
    for (j = 0; j < nsigs; j++)
        if (caught_sig[j])
            signal (sig[j], SIG_DFL);
#endif
}
}

static void
signal_init (void)
{
signal_setup (true);
}

static void
signal_restore (void)
{
signal_setup (false);
}

int
main (int argc, char **argv)
{
int i;
struct pending *thispend;
int n_files;

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

initialize_exit_failure (LS_FAILURE);
atexit (close_stdout);

static_assert (ARRAY_CARDINALITY (color_indicator) + 1
              == ARRAY_CARDINALITY (indicator_name));

exit_status = EXIT_SUCCESS;
print_dir_name = true;
pending_dirs = nullptr;

current_time.tv_sec = TYPE_MINIMUM (time_t);
current_time.tv_nsec = -1;

```

```

i = decode_switches (argc, argv);

if (print_with_color)
    parse_ls_color ();

/* Test print_with_color again, because the call to parse_ls_color
   may have just reset it -- e.g., if LS_COLORS is invalid. */

if (print_with_color)
{
    /* Don't use TAB characters in output. Some terminal
       emulators can't handle the combination of tabs and
       color codes on the same line. */
    tabsize = 0;
}

if (directories_first)
    check_symlink_mode = true;
else if (print_with_color)
{
    /* Avoid following symbolic links when possible. */
    if (is_colored (C_ORPHAN)
        || (is_colored (C_EXEC) && color_symlink_as_referent)
        || (is_colored (C_MISSING) && format == long_format))
        check_symlink_mode = true;
}

if (dereference == DEREF_UNDEFINED)
    dereference = ((immediate_dirs
                    || indicator_style == classify
                    || format == long_format)
                   ? DEREF_NEVER
                   : DEREF_COMMAND_LINE_SYMLINK_TO_DIR);

/* When using -R, initialize a data structure we'll use to
   detect any directory cycles. */
if (recursive)
{
    active_dir_set = hash_initialize (INITIAL_TABLE_SIZE, nullptr,
                                      dev_ino_hash,
                                      dev_ino_compare,
                                      dev_ino_free);
    if (active_dir_set == nullptr)
        xalloc_die ();
    obstack_init (&dev_ino_obstack);
}

localtz = tzalloc (getenv ("TZ"));

format_needs_stat = sort_type == sort_time || sort_type == sort_size
    || format == long_format
    || print_scontext
    || print_block_size;
format_needs_type = (! format_needs_stat
    && (recursive
        || print_with_color
        || indicator_style != none
        || directories_first));

```

```

if (dired)
{
    obstack_init (&dired_obstack);
    obstack_init (&subdirec_obstack);
}

if (print_hyperlink)
{
    file_escape_init ();

    hostname = xgethostname ();
    /* The hostname is generally ignored,
       so ignore failures obtaining it. */
    if (! hostname)
        hostname = "";
}

cwd_n_alloc = 100;
cwd_file = xnmalloc (cwd_n_alloc, sizeof * cwd_file);
cwd_n_used = 0;

clear_files ();

n_files = argc - i;

if (n_files <= 0)
{
    if (immediate_dirs)
        gobble_file (".", directory, NOT_AN_INODE_NUMBER, true, "");
    else
        queue_directory (".", nullptr, true);
}
else
    do
        gobble_file (argv[i++], unknown, NOT_AN_INODE_NUMBER, true, "");
    while (i < argc);

if (cwd_n_used)
{
    sort_files ();
    if (!immediate_dirs)
        extract_dirs_from_files (nullptr, true);
    /* 'cwd_n_used' might be zero now. */
}

/* In the following if/else blocks, it is sufficient to test 'pending_dirs'
   (and not pending_dirs->name) because there may be no markers in the queue
   at this point. A marker may be enqueued when extract_dirs_from_files is
   called with a non-empty string or via print_dir. */
if (cwd_n_used)
{
    print_current_files ();
    if (pending_dirs)
        dired_outbyte ('\n');
}
else if (n_files <= 1 && pending_dirs && pending_dirs->next == 0)
    print_dir_name = false;

while (pending_dirs)

```

```

{
thispend = pending_dirs;
pending_dirs = pending_dirs->next;

if (LOOP_DETECT)
{
    if (thispend->name == nullptr)
    {
        /* thispend->name == nullptr means this is a marker entry
           indicating we've finished processing the directory.
           Use its dev/ino numbers to remove the corresponding
           entry from the active_dir_set hash table. */
        struct dev_ino di = dev_ino_pop ();
        struct dev_ino *found = hash_remove (active_dir_set, &di);
        if (false)
            assert_matching_dev_ino (thispend->realname, di);
        affirm (found);
        dev_ino_free (found);
        free_pending_ent (thispend);
        continue;
    }
}

print_dir (thispend->name, thispend->realname,
           thispend->command_line_arg);

free_pending_ent (thispend);
print_dir_name = true;
}

if (print_with_color && used_color)
{
    int j;

    /* Skip the restore when it would be a no-op, i.e.,
       when left is "\033[" and right is "m". */
    if (!!(color_indicator[C_LEFT].len == 2
        && memcmp (color_indicator[C_LEFT].string, "\033[", 2) == 0
        && color_indicator[C_RIGHT].len == 1
        && color_indicator[C_RIGHT].string[0] == 'm'))
        restore_default_color ();

    fflush (stdout);

    signal_restore ();

    /* Act on any signals that arrived before the default was restored.
       This can process signals out of order, but there doesn't seem to
       be an easy way to do them in order, and the order isn't that
       important anyway. */
    for (j = stop_signal_count; j; j--)
        raise (SIGSTOP);
    j = interrupt_signal;
    if (j)
        raise (j);
}

if (dired)
{
    /* No need to free these since we're about to exit. */
}

```

```

dired_dump_obstack ("//DIRED//", &dired_obstack);
dired_dump_obstack ("//SUBDIRED//", &subdired_obstack);
printf ("//DIRED-OPTIONS// --quoting-style=%s\n",
       quoting_style_args[get_quoting_style (filename_quoting_options)]);
}

if (LOOP_DETECT)
{
    assure (hash_get_n_entries (active_dir_set) == 0);
    hash_free (active_dir_set);
}

return exit_status;
}

/* Return the line length indicated by the value given by SPEC, or -1
if unsuccessful. 0 means no limit on line length. */

static ptrdiff_t
decode_line_length (char const *spec)
{
    uintmax_t val;

    /* Treat too-large values as if they were 0, which is
       effectively infinity. */
    switch (xstrtoumax (spec, nullptr, 0, &val, ""))
    {
        case LONGINT_OK:
            return val <= MIN (PTRDIFF_MAX, SIZE_MAX) ? val : 0;

        case LONGINT_OVERFLOW:
            return 0;

        default:
            return -1;
    }
}

/* Return true if standard output is a tty, caching the result. */

static bool
stdout_isatty (void)
{
    static signed char out_tty = -1;
    if (out_tty < 0)
        out_tty = isatty (STDOUT_FILENO);
    assume (out_tty == 0 || out_tty == 1);
    return out_tty;
}

/* Set all the option flags according to the switches specified.
Return the index of the first non-option argument. */

static int
decode_switches (int argc, char **argv)
{
    char const *time_style_option = nullptr;

    /* These variables are false or -1 unless a switch says otherwise. */
    bool kibibytes_specified = false;
}

```

```

int format_opt = -1;
int hide_control_chars_opt = -1;
int quoting_style_opt = -1;
int sort_opt = -1;
ptrdiff_t tabsize_opt = -1;
ptrdiff_t width_opt = -1;

while (true)
{
    int oi = -1;
    int c = getopt_long (argc, argv,
                         "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZLNQRSTUXZ1",
                         long_options, &oi);
    if (c == -1)
        break;

    switch (c)
    {
        case 'a':
            ignore_mode = IGNORE_MINIMAL;
            break;

        case 'b':
            quoting_style_opt = escape_quoting_style;
            break;

        case 'c':
            time_type = time_ctime;
            break;

        case 'd':
            immediate_dirs = true;
            break;

        case 'f':
            ignore_mode = IGNORE_MINIMAL; /* enable -a */
            sort_opt = sort_none; /* enable -U */
            if (format_opt == long_format)
                format_opt = -1; /* disable -l */
            print_with_color = false; /* disable --color */
            print_hyperlink = false; /* disable --hyperlink */
            print_block_size = false; /* disable -s */
            break;

        case FILE_TYPE_INDICATOR_OPTION: /* --file-type */
            indicator_style = file_type;
            break;

        case 'g':
            format_opt = long_format;
            print_owner = false;
            break;

        case 'h':
            file_human_output_opts = human_output_opts =
                human_autoscale | human_SI | human_base_1024;
            file_output_block_size = output_block_size = 1;
            break;

        case 'i':
    }
}

```

```
print_inode = true;
break;

case 'k':
kibibytes_specified = true;
break;

case 'l':
format_opt = long_format;
break;

case 'm':
format_opt = with_commas;
break;

case 'n':
numeric_ids = true;
format_opt = long_format;
break;

case 'o': /* Just like -l, but don't display group info. */
format_opt = long_format;
print_group = false;
break;

case 'p':
indicator_style = slash;
break;

case 'q':
hide_control_chars_opt = true;
break;

case 'r':
sort_reverse = true;
break;

case 's':
print_block_size = true;
break;

case 't':
sort_opt = sort_time;
break;

case 'u':
time_type = time_atime;
break;

case 'v':
sort_opt = sort_version;
break;

case 'w':
width_opt = decode_line_length (optarg);
if (width_opt < 0)
    error (LS_FAILURE, 0, "%s: %s", _("invalid line width"),
          quote (optarg));
break;
```

```

case 'x':
format_opt = horizontal;
break;

case 'A':
ignore_mode = IGNORE_DOT_AND_DOTDOT;
break;

case 'B':
add_ignore_pattern ("*~");
add_ignore_pattern (".*~");
break;

case 'C':
format_opt = many_per_line;
break;

case 'D':
format_opt = long_format;
print_hyperlink = false;
dired = true;
break;

case 'F':
{
    int i;
    if (optarg)
        i = XARGMATCH ("--classify", optarg, when_args, when_types);
    else
        /* Using --classify with no argument is equivalent to using
           --classify=always. */
        i = when_always;

    if (i == when_always || (i == when_if_tty && stdout_isatty ()))
        indicator_style = classify;
    break;
}

case 'G':          /* inhibit display of group info */
print_group = false;
break;

case 'H':
dereference = DEREF_COMMAND_LINE_ARGUMENTS;
break;

case DEREference_COMMAND_LINE_SYMLINK_TO_DIR_OPTION:
dereference = DEREF_COMMAND_LINE_SYMLINK_TO_DIR;
break;

case 'I':
add_ignore_pattern (optarg);
break;

case 'L':
dereference = DEREF_ALWAYS;
break;

case 'N':
quoting_style_opt = literal_quoting_style;

```

```

break;

case 'Q':
quoting_style_opt = c_quoting_style;
break;

case 'R':
recursive = true;
break;

case 'S':
sort_opt = sort_size;
break;

case 'T':
tabsize_opt = xnumtoumax (optarg, 0, 0, MIN (PTRDIFF_MAX, SIZE_MAX),
                           "", _("invalid tab size"), LS_FAILURE);
break;

case 'U':
sort_opt = sort_none;
break;

case 'X':
sort_opt = sort_extension;
break;

case '1':
/* -1 has no effect after -l. */
if (format_opt != long_format)
    format_opt = one_per_line;
break;

case AUTHOR_OPTION:
print_author = true;
break;

case HIDE_OPTION:
{
    struct ignore_pattern *hide = xmalloc (sizeof *hide);
    hide->pattern = optarg;
    hide->next = hide_patterns;
    hide_patterns = hide;
}
break;

case SORT_OPTION:
sort_opt = XARGMATCH ("--sort", optarg, sort_args, sort_types);
break;

case GROUP_DIRECTORIES_FIRST_OPTION:
directories_first = true;
break;

case TIME_OPTION:
time_type = XARGMATCH ("--time", optarg, time_args, time_types);
break;

case FORMAT_OPTION:
format_opt = XARGMATCH ("--format", optarg, format_args,

```

```

        format_types);
break;

case FULL_TIME_OPTION:
format_opt = long_format;
time_style_option = "full-iso";
break;

case COLOR_OPTION:
{
    int i;
    if (optarg)
i = XARGMATCH ("--color", optarg, when_args, when_types);
else
/* Using --color with no argument is equivalent to using
   --color=always. */
i = when_always;

print_with_color = (i == when_always
                    || (i == when_if_tty && stdout_isatty ()) );
break;
}

case HYPERLINK_OPTION:
{
    int i;
    if (optarg)
i = XARGMATCH ("--hyperlink", optarg, when_args, when_types);
else
/* Using --hyperlink with no argument is equivalent to using
   --hyperlink=always. */
i = when_always;

print_hyperlink = (i == when_always
                    || (i == when_if_tty && stdout_isatty ()) );
break;
}

case INDICATOR_STYLE_OPTION:
indicator_style = XARGMATCH ("--indicator-style", optarg,
                             indicator_style_args,
                             indicator_style_types);
break;

case QUOTING_STYLE_OPTION:
quoting_style_opt = XARGMATCH ("--quoting-style", optarg,
                               quoting_style_args,
                               quoting_style_vals);
break;

case TIME_STYLE_OPTION:
time_style_option = optarg;
break;

case SHOW_CONTROL_CHARS_OPTION:
hide_control_chars_opt = false;
break;

case BLOCK_SIZE_OPTION:
{

```

```

        enum strtol_error e = human_options (optarg, &human_output_opts,
                                             &output_block_size);
        if (e != LONGINT_OK)
            xstrtol_fatal (e, oi, 0, long_options, optarg);
        file_human_output_opts = human_output_opts;
        file_output_block_size = output_block_size;
    }
    break;

    case SI_OPTION:
        file_human_output_opts = human_output_opts =
            human_autoscale | human_SI;
        file_output_block_size = output_block_size = 1;
    break;

    case 'Z':
        print_scontext = true;
    break;

    case ZERO_OPTION:
        eolbyte = 0;
        hide_control_chars_opt = false;
        if (format_opt != long_format)
            format_opt = one_per_line;
        print_with_color = false;
        quoting_style_opt = literal_quoting_style;
    break;

    case_GETOPT_HELP_CHAR;

    case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

    default:
        usage (LS_FAILURE);
    }
}

if (!output_block_size)
{
    char const *ls_block_size = getenv ("LS_BLOCK_SIZE");
    human_options (ls_block_size,
                   &human_output_opts, &output_block_size);
    if (ls_block_size || getenv ("BLOCK_SIZE"))
    {
        file_human_output_opts = human_output_opts;
        file_output_block_size = output_block_size;
    }
    if (kibibytes_specified)
    {
        human_output_opts = 0;
        output_block_size = 1024;
    }
}

format = (0 <= format_opt ? format_opt
    : ls_mode == LS_LS ? (stdout_isatty ()
        ? many_per_line : one_per_line)
    : ls_mode == LS_MULTI_COL ? many_per_line
    : /* ls_mode == LS_LONG_FORMAT */ long_format);

```

```

/* If the line length was not set by a switch but is needed to determine
   output, go to the work of obtaining it from the environment. */
ptrdiff_t linelen = width_opt;
if (format == many_per_line || format == horizontal || format == with_commas
    || print_with_color)
{
#endif TIOCGWINSZ
    if (linelen < 0)
    {
        struct winsize ws;
        if (stdout_isatty ())
            && 0 <= ioctl (STDOUT_FILENO, TIOCGWINSZ, &ws)
            && 0 < ws.ws_col
        linelen = ws.ws_col <= MIN (PTRDIFF_MAX, SIZE_MAX) ? ws.ws_col : 0;
    }
#endif
    if (linelen < 0)
    {
        char const *p = getenv ("COLUMNS");
        if (p && *p)
        {
            linelen = decode_line_length (p);
            if (linelen < 0)
                error (0, 0,
                    _("ignoring invalid width"
                      " in environment variable COLUMNS: %s"),
                    quote (p));
        }
    }
}

line_length = linelen < 0 ? 80 : linelen;

/* Determine the max possible number of display columns. */
max_idx = line_length / MIN_COLUMN_WIDTH;
/* Account for first display column not having a separator,
   or line_lengths shorter than MIN_COLUMN_WIDTH. */
max_idx += line_length % MIN_COLUMN_WIDTH != 0;

if (format == many_per_line || format == horizontal || format == with_commas)
{
    if (0 <= tabsize_opt)
        tabsize = tabsize_opt;
    else
    {
        tabsize = 8;
        char const *p = getenv ("TABSIZE");
        if (p)
        {
            uintmax_t tmp;
            if (xstrtoumax (p, nullptr, 0, &tmp, "") == LONGINT_OK
                && tmp <= SIZE_MAX)
                tabsize = tmp;
            else
                error (0, 0,
                    _("ignoring invalid tab size"
                      " in environment variable TABSIZE: %s"),
                    quote (p));
        }
    }
}

```

```

}

qmark_funny_chars = (hide_control_chars_opt < 0
    ? ls_mode == LS_LS && stdout_isatty ()
    : hide_control_chars_opt);

int qs = quoting_style_opt;
if (qs < 0)
    qs = getenv_quoting_style ();
if (qs < 0)
    qs = (ls_mode == LS_LS
        ? (stdout_isatty () ? shell_escape_quoting_style : -1)
        : escape_quoting_style);
if (0 <= qs)
    set_quoting_style (nullptr, qs);
qs = get_quoting_style (nullptr);
align_variable_outer_quotes
    = ((format == long_format
        || ((format == many_per_line || format == horizontal) && line_length))
    && (qs == shell_quoting_style
        || qs == shell_escape_quoting_style
        || qs == c_maybe_quoting_style));
filename_quoting_options = clone_quoting_options (nullptr);
if (qs == escape_quoting_style)
    set_char_quoting (filename_quoting_options, ' ', 1);
if (file_type <= indicator_style)
{
    char const *p;
    for (p = &"*=>@|[indicator_style - file_type]; *p; p++)
        set_char_quoting (filename_quoting_options, *p, 1);
}

dirname_quoting_options = clone_quoting_options (nullptr);
set_char_quoting (dirname_quoting_options, ':', 1);

/* --dired implies --format=long (-l) and sans --hyperlink.
   So ignore it if those overridden. */
dired &= (format == long_format) & !print_hyperlink;

if (eolbyte < dired)
    error (LS_FAILURE, 0, _("--dired and --zero are incompatible"));

/* If -c or -u is specified and not -l (or any other option that implies -l),
   and no sort-type was specified, then sort by the ctime (-c) or atime (-u).
   The behavior of ls when using either -c or -u but with neither -l nor -t
   appears to be unspecified by POSIX. So, with GNU ls, '-u' alone means
   sort by atime (this is the one that's not specified by the POSIX spec),
   -lu means show atime and sort by name, -lut means show atime and sort
   by atime. */
sort_type = (0 <= sort_opt ? sort_opt
    : (format != long_format
        && (time_type == time_ctime || time_type == time_atime
            || time_type == time_btime))
    ? sort_time : sort_name);

if (format == long_format)
{
    char const *style = time_style_option;
    static char const posix_prefix[] = "posix-";
}

```

```

if (! style)
{
    style = getenv ("TIME_STYLE");
    if (! style)
        style = "locale";
}

while (STREQ_LEN (style, posix_prefix, sizeof posix_prefix - 1))
{
    if (! hard_locale (LC_TIME))
        return optind;
    style += sizeof posix_prefix - 1;
}

if (*style == '+')
{
    char const *p0 = style + 1;
    char *p0nl = strchr (p0, '\n');
    char const *p1 = p0;
    if (p0nl)
    {
        if (strchr (p0nl + 1, '\n'))
            error (LS_FAILURE, 0, _("invalid time style format %s"),
                   quote (p0));
        *p0nl++ = '\0';
        p1 = p0nl;
    }
    long_time_format[0] = p0;
    long_time_format[1] = p1;
}
else
{
    ptrdiff_t res = argmatch (style, time_style_args,
                               (char const *) time_style_types,
                               sizeof (*time_style_types));
    if (res < 0)
    {
        /* This whole block used to be a simple use of XARGMATCH.
           but that didn't print the "posix-"-prefixed variants or
           the "+"-prefixed format string option upon failure. */
        argmatch_invalid ("time style", style, res);

        /* The following is a manual expansion of argmatch_valid,
           but with the added "+ ..." description and the [posix-]
           prefixes prepended. Note that this simplification works
           only because all four existing time_style_types values
           are distinct. */
        fputs (_("Valid arguments are:\n"), stderr);
        char const *const *p = time_style_args;
        while (*p)
            fprintf (stderr, " - [posix-]%s\n", *p++);
        fputs (_(" - +FORMAT (e.g., +%H:%M) for a 'date'-style"
                " format\n"), stderr);
        usage (LS_FAILURE);
    }
    switch (res)
    {
        case full_iso_time_style:
            long_time_format[0] = long_time_format[1] =

```

```

        "%Y-%m-%d %H:%M:%S.%N %z";
    break;

    case long_iso_time_style:
long_time_format[0] = long_time_format[1] = "%Y-%m-%d %H:%M";
break;

    case iso_time_style:
long_time_format[0] = "%Y-%m-%d ";
long_time_format[1] = "%m-%d %H:%M";
break;

    case locale_time_style:
if (hard_locale (LC_TIME))
{
    for (int i = 0; i < 2; i++)
        long_time_format[i] =
            dcgettext (nullptr, long_time_format[i], LC_TIME);
}
}

abformat_init ();
}

return optind;
}

```

/\* Parse a string as part of the LS\_COLORS variable; this may involve decoding all kinds of escape characters. If equals\_end is set an unescaped equal sign ends the string, otherwise only a : or \0 does. Set \*OUTPUT\_COUNT to the number of bytes output. Return true if successful.

The resulting string is *\*not\** null-terminated, but may contain embedded nulls.

Note that both dest and src are char \*\*; on return they point to the first free byte after the array and the character that ended the input string, respectively. \*/

```

static bool
get_funky_string (char **dest, char const **src, bool equals_end,
                  size_t *output_count)
{
char num;                      /* For numerical codes */
size_t count;                   /* Something to count with */
enum {
    ST_GND, ST_BACKSLASH, ST_OCTAL, ST_HEX, ST_CARET, ST_END, ST_ERROR
} state;
char const *p;
char *q;

p = *src;                      /* We don't want to double-indirect */
q = *dest;                     /* the whole darn time. */

count = 0;                      /* No characters counted in yet. */
num = 0;

state = ST_GND;                 /* Start in ground state. */

```

```

while (state < ST_END)
{
    switch (state)
    {
        case ST_GND:           /* Ground state (no escapes) */
            switch (*p)
            {
                case ':':
                case '\0':
                    state = ST_END; /* End of string */
                    break;
                case '\\':
                    state = ST_BACKSLASH; /* Backslash escape sequence */
                    ++p;
                    break;
                case '^':
                    state = ST_CARET; /* Caret escape */
                    ++p;
                    break;
                case '=':
                    if (equals_end)
                    {
                        state = ST_END; /* End */
                        break;
                    }
                    FALLTHROUGH;
                default:
                    *(q++) = *(p++);
                    ++count;
                    break;
            }
            break;

        case ST_BACKSLASH: /* Backslash escaped character */
            switch (*p)
            {
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                    state = ST_OCTAL; /* Octal sequence */
                    num = *p - '0';
                    break;
                case 'x':
                case 'X':
                    state = ST_HEX; /* Hex sequence */
                    num = 0;
                    break;
                case 'a':           /* Bell */
                    num = '\a';
                    break;
                case 'b':           /* Backspace */
                    num = '\b';
                    break;
                case 'e':           /* Escape */
                    num = 27;

```

```

break;
case 'f':           /* Form feed */
num = '\f';
break;
case 'n':           /* Newline */
num = '\n';
break;
case 'r':           /* Carriage return */
num = '\r';
break;
case 't':           /* Tab */
num = '\t';
break;
case 'v':           /* Vtab */
num = '\v';
break;
case '?':           /* Delete */
num = 127;
break;
case '_':           /* Space */
num = ' ';
break;
case '\0':          /* End of string */
state = ST_ERROR; /* Error! */
break;
default:            /* Escaped character like \ ^ : = */
num = *p;
break;
}
if (state == ST_BACKSLASH)
{
*(q++) = num;
++count;
state = ST_GND;
}
++p;
break;

case ST_OCTAL:        /* Octal sequence */
if (*p < '0' || *p > '7')
{
*(q++) = num;
++count;
state = ST_GND;
}
else
    num = (num << 3) + (*p++) - '0';
break;

case ST_HEX:          /* Hex sequence */
switch (*p)
{
case '0':
case '1':
case '2':
case '3':
case '4':
case '5':
case '6':
case '7':

```

```

        case '8':
        case '9':
            num = (num << 4) + (*p++ - '0');
            break;
        case 'a':
        case 'b':
        case 'c':
        case 'd':
        case 'e':
        case 'f':
            num = (num << 4) + (*p++ - 'a') + 10;
            break;
        case 'A':
        case 'B':
        case 'C':
        case 'D':
        case 'E':
        case 'F':
            num = (num << 4) + (*p++ - 'A') + 10;
            break;
        default:
            *(q++) = num;
            ++count;
            state = ST_GND;
            break;
    }
break;

case ST_CARET:          /* Caret escape */
state = ST_GND;          /* Should be the next state... */
if (*p >= '@' && *p <= '~')
{
    *(q++) = *(p++) & 037;
    ++count;
}
else if (*p == '?')
{
    *(q++) = 127;
    ++count;
}
else
    state = ST_ERROR;
break;

default:
unreachable ();
}

}

*dest = q;
*src = p;
*output_count = count;

return state != ST_ERROR;
}

enum parse_state
{
    PS_START = 1,
    PS_2,

```

```

PS_3,
PS_4,
PS_DONE,
PS_FAIL
};

/* Check if the content of TERM is a valid name in dircolors. */

static bool
known_term_type (void)
{
char const *term = getenv ("TERM");
if (! term || ! *term)
    return false;

char const *line = G_line;
while (line - G_line < sizeof (G_line))
{
    if (STRNCMP_LIT (line, "TERM ") == 0)
    {
        if (fnmatch (line + 5, term, 0) == 0)
            return true;
    }
    line += strlen (line) + 1;
}

return false;
}

static void
parse_ls_color (void)
{
char const *p;          /* Pointer to character being parsed */
char *buf;              /* color_buf buffer pointer */
int ind_no;             /* Indicator number */
char label[3];          /* Indicator label */
struct color_ext_type *ext; /* Extension we are working on */

if ((p = getenv ("LS_COLORS")) == nullptr || *p == '\0')
{
    /* LS_COLORS takes precedence, but if that's not set then
       honor the COLORTERM and TERM env variables so that
       we only go with the internal ANSI color codes if the
       former is non empty or the latter is set to a known value. */
    char const *colorterm = getenv ("COLORTERM");
    if (! (colorterm && *colorterm) && ! known_term_type ())
        print_with_color = false;
    return;
}

ext = nullptr;
strcpy (label, "??");

/* This is an overly conservative estimate, but any possible
   LS_COLORS string will *not* generate a color_buf longer than
   itself, so it is a safe way of allocating a buffer in
   advance. */
buf = color_buf = xstrdup (p);

```

```

enum parse_state state = PS_START;
while (true)
{
    switch (state)
    {
        case PS_START:           /* First label character */
            switch (*p)
            {
                case ':':
                    ++p;
                    break;

                case '*':
                    /* Allocate new extension block and add to head of
                     linked list (this way a later definition will
                     override an earlier one, which can be useful for
                     having terminal-specific defs override global). */

                    ext = xmalloc (sizeof *ext);
                    ext->next = color_ext_list;
                    color_ext_list = ext;
                    ext->exact_match = false;

                    ++p;
                    ext->ext.string = buf;

                    state = (get_funky_string (&buf, &p, true, &ext->ext.len)
                            ? PS_4 : PS_FAIL);
                    break;

                case '\0':
                    state = PS_DONE;    /* Done! */
                    goto done;

                default:              /* Assume it is file type label */
                    label[0] = *(p++);
                    state = PS_2;
                    break;
            }
            break;

        case PS_2:               /* Second label character */
            if (*p)
            {
                label[1] = *(p++);
                state = PS_3;
            }
            else
                state = PS_FAIL;    /* Error */
            break;

        case PS_3:               /* Equal sign after indicator label */
            state = PS_FAIL; /* Assume failure... */
            if (*(p++) == '=')/* It *should* be... */
            {
                for (ind_no = 0; indicator_name[ind_no] != nullptr; ++ind_no)
                {
                    if (STREQ (label, indicator_name[ind_no]))
                    {
                        color_indicator[ind_no].string = buf;

```

```

state = (get_funky_string (&buf, &p, false,
                           &color_indicator[ind_no].len)
         ? PS_START : PS_FAIL);
break;
}
}
if (state == PS_FAIL)
    error (0, 0, _("unrecognized prefix: %s"), quote (label));
}
break;

case PS_4:           /* Equal sign after *.ext */
if (*(p++) == '=')
{
    ext->seq.string = buf;
    state = (get_funky_string (&buf, &p, false, &ext->seq.len)
              ? PS_START : PS_FAIL);
}
else
    state = PS_FAIL;
break;

case PS_FAIL:
goto done;

default:
affirm (false);
}
}
done:

if (state == PS_FAIL)
{
struct color_ext_type *e;
struct color_ext_type *e2;

error (0, 0,
       _("unparsable value for LS_COLORS environment variable"));
free (color_buf);
for (e = color_ext_list; e != nullptr; /* empty */)
{
    e2 = e;
    e = e->next;
    free (e2);
}
print_with_color = false;
}
else
{
/* Postprocess list to set EXACT_MATCH on entries where there are
   different cased extensions with separate sequences defined.
   Also set ext.len to SIZE_MAX on any entries that can't
   match due to precedence, to avoid redundant string compares. */
struct color_ext_type *e1;

for (e1 = color_ext_list; e1 != nullptr; e1 = e1->next)
{
    struct color_ext_type *e2;
    bool case_ignored = false;

```

```

for (e2 = e1->next; e2 != nullptr; e2 = e2->next)
{
    if (e2->ext.len < SIZE_MAX && e1->ext.len == e2->ext.len)
    {
        if (memcmp (e1->ext.string, e2->ext.string, e1->ext.len) == 0)
            e2->ext.len = SIZE_MAX; /* Ignore */
        else if (c_strncasecmp (e1->ext.string, e2->ext.string,
                               e1->ext.len) == 0)
        {
            if (case_ignored)
            {
                e2->ext.len = SIZE_MAX; /* Ignore */
            }
            else if (e1->seq.len == e2->seq.len
                      && memcmp (e1->seq.string, e2->seq.string,
                                 e1->seq.len) == 0)
            {
                e2->ext.len = SIZE_MAX; /* Ignore */
                case_ignored = true; /* Ignore all subsequent */
            }
            else
            {
                e1->exact_match = true;
                e2->exact_match = true;
            }
        }
    }
}
}

if (color_indicator[C_LINK].len == 6
    && !STRNCMP_LIT (color_indicator[C_LINK].string, "target"))
    color_symlink_as_referent = true;
}

/* Return the quoting style specified by the environment variable
QUOTING_STYLE if set and valid, -1 otherwise. */

static int
getenv_quoting_style (void)
{
char const *q_style = getenv ("QUOTING_STYLE");
if (!q_style)
    return -1;
int i = ARGMATCH (q_style, quoting_style_args, quoting_style_vals);
if (i < 0)
{
    error (0, 0,
           ("ignoring invalid value"
            " of environment variable QUOTING_STYLE: %s"),
            quote (q_style));
    return -1;
}
return quoting_style_vals[i];
}

/* Set the exit status to report a failure. If SERIOUS, it is a
serious failure; otherwise, it is merely a minor problem. */

```

```

static void
set_exit_status (bool serious)
{
if (serious)
    exit_status = LS_FAILURE;
else if (exit_status == EXIT_SUCCESS)
    exit_status = LS_MINOR_PROBLEM;
}

/* Assuming a failure is serious if SERIOUS, use the printf-style
MESSAGE to report the failure to access a file named FILE. Assume
errno is set appropriately for the failure. */

static void
file_failure (bool serious, char const *message, char const *file)
{
error (0, errno, message, quoteaf (file));
set_exit_status (serious);
}

/* Request that the directory named NAME have its contents listed later.
If REALNAME is nonzero, it will be used instead of NAME when the
directory name is printed. This allows symbolic links to directories
to be treated as regular directories but still be listed under their
real names. NAME == nullptr is used to insert a marker entry for the
directory named in REALNAME.
If NAME is non-null, we use its dev/ino information to save
a call to stat -- when doing a recursive (-R) traversal.
COMMAND_LINE_ARG means this directory was mentioned on the command line. */

static void
queue_directory (char const *name, char const *realname, bool command_line_arg)
{
struct pending *new = xmalloc (sizeof *new);
new->realname = realname ? xstrdup (realname) : nullptr;
new->name = name ? xstrdup (name) : nullptr;
new->command_line_arg = command_line_arg;
new->next = pending_dirs;
pending_dirs = new;
}

/* Read directory NAME, and list the files in it.
If REALNAME is nonzero, print its name instead of NAME;
this is used for symbolic links to directories.
COMMAND_LINE_ARG means this directory was mentioned on the command line. */

static void
print_dir (char const *name, char const *realname, bool command_line_arg)
{
DIR *dirp;
struct dirent *next;
uintmax_t total_blocks = 0;
static bool first = true;

errno = 0;
dirp = opendir (name);
if (!dirp)
{
    file_failure (command_line_arg, _("cannot open directory %s"), name);
    return;
}

```

```

    }

if (LOOP_DETECT)
{
    struct stat dir_stat;
    int fd = dirfd (dirp);

/* If dirfd failed, endure the overhead of stat'ing by path */
if ((0 <= fd
    ? fstat_for_ino (fd, &dir_stat)
    : stat_for_ino (name, &dir_stat)) < 0)
{
    file_failure (command_line_arg,
                  _("cannot determine device and inode of %s"), name);
    closedir (dirp);
    return;
}

/* If we've already visited this dev/inode pair, warn that
   we've found a loop, and do not process this directory. */
if (visit_dir (dir_stat.st_dev, dir_stat.st_ino))
{
    error (0, 0, _("%s: not listing already-listed directory"),
           quotef (name));
    closedir (dirp);
    set_exit_status (true);
    return;
}

dev_ino_push (dir_stat.st_dev, dir_stat.st_ino);
}

clear_files ();

if (recursive || print_dir_name)
{
    if (!first)
        dired_outbyte ('\n');
    first = false;
    dired_indent ();

    char *absolute_name = nullptr;
    if (print_hyperlink)
    {
        absolute_name = canonicalize_filename_mode (name, CAN_MISSING);
        if (! absolute_name)
            file_failure (command_line_arg,
                          _("error canonicalizing %s"), name);
    }
    quote_name (realname ? realname : name, dirname_quoting_options, -1,
                nullptr, true, &subdirec_obstack, absolute_name);

    free (absolute_name);

    dired_outstring (" :\n");
}

/* Read the directory entries, and insert the subfiles into the 'cwd_file'
   table. */

```

```

while (true)
{
    /* Set errno to zero so we can distinguish between a readdir failure
       and when readdir simply finds that there are no more entries. */
    errno = 0;
    next = readdir (dirp);
    /* Some readdir()s do not absorb ENOENT (dir deleted but open). */
    if (errno == ENOENT)
        errno = 0;
    if (next)
    {
        if (! file_ignored (next->d_name))
        {
            enum filetype type = unknown;

#if HAVE_STRUCT_DIRENT_D_TYPE
            switch (next->d_type)
            {
                case DT_BLK: type = blockdev; break;
                case DT_CHR: type = chardev; break;
                case DT_DIR: type = directory; break;
                case DT_FIFO: type = fifo; break;
                case DT_LNK: type = symbolic_link; break;
                case DT_REG: type = normal; break;
                case DT_SOCK: type = sock; break;
# ifdef DT_WHT
                case DT_WHT: type = whiteout; break;
# endif
            }
#endif
            total_blocks += gobble_file (next->d_name, type,
                                         RELIABLE_D_INO (next),
                                         false, name);

            /* In this narrow case, print out each name right away, so
               ls uses constant memory while processing the entries of
               this directory. Useful when there are many (millions)
               of entries in a directory. */
            if (format == one_per_line && sort_type == sort_none
                && !print_block_size && !recursive)
            {
                /* We must call sort_files in spite of
                   "sort_type == sort_none" for its initialization
                   of the sorted_file vector. */
                sort_files ();
                print_current_files ();
                clear_files ();
            }
        }
    }
    else if (errno != 0)
    {
        file_failure (command_line_arg, _("reading directory %s"), name);
        if (errno != EVERFLOW)
            break;
    }
else
    break;

/* When processing a very large directory, and since we've inhibited

```

```

interrupts, this loop would take so long that ls would be annoyingly
uninterruptible. This ensures that it handles signals promptly. */
process_signals ();
}

if (closedir (dirp) != 0)
{
    file_failure (command_line_arg, _("closing directory %s"), name);
    /* Don't return; print whatever we got. */
}

/* Sort the directory contents. */
sort_files ();

/* If any member files are subdirectories, perhaps they should have their
contents listed rather than being mentioned here as files. */

if (recursive)
    extract_dirs_from_files (name, false);

if (format == long_format || print_block_size)
{
    char buf[LONGEST_HUMAN_READABLE + 3];
    char *p = human_readable (total_blocks, buf + 1, human_output_opts,
                             ST_NBLOCKSIZE, output_block_size);
    char *pend = p + strlen (p);
    *--p = ' ';
    *pend++ = eolbyte;
    dirent_indent ();
    dirent_outstring (_("total"));
    dirent_outbuf (p, pend - p);
}

if (cwd_n_used)
    print_current_files ();
}

/* Add 'pattern' to the list of patterns for which files that match are
not listed. */

static void
add_ignore_pattern (char const *pattern)
{
struct ignore_pattern *ignore;

ignore = xmalloc (sizeof *ignore);
ignore->pattern = pattern;
/* Add it to the head of the linked list. */
ignore->next = ignore_patterns;
ignore_patterns = ignore;
}

/* Return true if one of the PATTERNS matches FILE. */

static bool
patterns_match (struct ignore_pattern const *patterns, char const *file)
{
struct ignore_pattern const *p;
for (p = patterns; p; p = p->next)
    if (fnmatch (p->pattern, file, FNM_PERIOD) == 0)

```

```

        return true;
    return false;
}

/* Return true if FILE should be ignored. */

static bool
file_ignored (char const *name)
{
    return ((ignore_mode != IGNORE_MINIMAL
        && name[0] == '.'
        && (ignore_mode == IGNORE_DEFAULT || ! name[1 + (name[1] == '.')]))
        || (ignore_mode == IGNORE_DEFAULT
            && patterns_match (hide_patterns, name))
        || patterns_match (ignore_patterns, name));
}

/* POSIX requires that a file size be printed without a sign, even
   when negative. Assume the typical case where negative sizes are
   actually positive values that have wrapped around. */

static uintmax_t
unsigned_file_size (off_t size)
{
    return size + (size < 0) * ((uintmax_t) OFF_T_MAX - OFF_T_MIN + 1);
}

#ifndef HAVE_CAP
/* Return true if NAME has a capability (see linux/capability.h) */
static bool
has_capability (char const *name)
{
    char *result;
    bool has_cap;

    cap_t cap_d = cap_get_file (name);
    if (cap_d == nullptr)
        return false;

    result = cap_to_text (cap_d, nullptr);
    cap_free (cap_d);
    if (!result)
        return false;

    /* check if human-readable capability string is empty */
    has_cap = !!*result;

    cap_free (result);
    return has_cap;
}
#else
static bool
has_capability (MAYBE_UNUSED char const *name)
{
    errno = ENOTSUP;
    return false;
}
#endif

/* Enter and remove entries in the table 'cwd_file'. */

```

```

static void
free_ent (struct fileinfo *f)
{
    free (f->name);
    free (f->linkname);
    free (f->absolute_name);
    if (f->scontext != UNKNOWN_SECURITY_CONTEXT)
        {
            if (is_smack_enabled ())
                free (f->scontext);
            else
                freecon (f->scontext);
        }
}

/* Empty the table of files. */
static void
clear_files (void)
{
    for (size_t i = 0; i < cwd_n_used; i++)
    {
        struct fileinfo *f = sorted_file[i];
        free_ent (f);
    }

    cwd_n_used = 0;
    cwd_some_quoted = false;
    any_has_acl = false;
    inode_number_width = 0;
    block_size_width = 0;
    nlink_width = 0;
    owner_width = 0;
    group_width = 0;
    author_width = 0;
    scontext_width = 0;
    major_device_number_width = 0;
    minor_device_number_width = 0;
    file_size_width = 0;
}

/* Return true if ERR implies lack-of-support failure by a
getxattr-calling function like getfilecon or file_has_acl. */
static bool
errno_unsupported (int err)
{
    return (err == EINVAL || err == ENOSYS || is_ENOTSUP (err));
}

/* Cache *getfilecon failure, when it's trivial to do so.
Like getfilecon/lgetfilecon, but when F's st_dev says it's doesn't
support getting the security context, fail with ENOTSUP immediately. */
static int
getfilecon_cache (char const *file, struct fileinfo *f, bool deref)
{
    /* st_dev of the most recently processed device for which we've
       found that []getfilecon fails indicating lack of support. */
    static dev_t unsupported_device;

    if (f->stat.st_dev == unsupported_device)

```

```

{
    errno = ENOTSUP;
    return -1;
}
int r = 0;
#ifndef HAVE_SMACK
if (is_smack_enabled ())
    r = smack_new_label_from_path (file, "security.SMACK64", deref,
                                  &f->scontext);
else
#endif
r = (deref
     ? getfilecon (file, &f->scontext)
     : lgetfilecon (file, &f->scontext));
if (r < 0 && errno_unsupported (errno))
    unsupported_device = f->stat.st_dev;
return r;
}

/* Cache file_has_acl failure, when it's trivial to do.
Like file_has_acl, but when F's st_dev says it's on a file
system lacking ACL support, return 0 with ENOTSUP immediately. */
static int
file_has_acl_cache (char const *file, struct fileinfo *f)
{
/* st_dev of the most recently processed device for which we've
   found that file_has_acl fails indicating lack of support. */
static dev_t unsupported_device;

if (f->stat.st_dev == unsupported_device)
{
    errno = ENOTSUP;
    return 0;
}

/* Zero errno so that we can distinguish between two 0-returning cases:
   "has-ACL-support, but only a default ACL" and "no ACL support". */
errno = 0;
int n = file_has_acl (file, &f->stat);
if (n <= 0 && errno_unsupported (errno))
    unsupported_device = f->stat.st_dev;
return n;
}

/* Cache has_capability failure, when it's trivial to do.
Like has_capability, but when F's st_dev says it's on a file
system lacking capability support, return 0 with ENOTSUP immediately. */
static bool
has_capability_cache (char const *file, struct fileinfo *f)
{
/* st_dev of the most recently processed device for which we've
   found that has_capability fails indicating lack of support. */
static dev_t unsupported_device;

if (f->stat.st_dev == unsupported_device)
{
    errno = ENOTSUP;
    return 0;
}

```

```

bool b = has_capability (file);
if ( !b && errno_unsupported (errno))
    unsupported_device = f->stat.st_dev;
return b;
}

static bool
needs_quoting (char const *name)
{
char test[2];
size_t len = quotearg_buffer (test, sizeof test , name, -1,
                             filename_quoting_options);
return *name != *test || strlen (name) != len;
}

/* Add a file to the current table of files.
Verify that the file exists, and print an error message if it does not.
Return the number of blocks that the file occupies. */
static uintmax_t
gobble_file (char const *name, enum filetype type, ino_t inode,
             bool command_line_arg, char const *dirname)
{
uintmax_t blocks = 0;
struct fileinfo *f;

/* An inode value prior to gobble_file necessarily came from readdir,
   which is not used for command line arguments. */
affirm (! command_line_arg || inode == NOT_AN_INODE_NUMBER);

if (cwd_n_used == cwd_n_alloc)
{
    cwd_file = xnrealloc (cwd_file, cwd_n_alloc, 2 * sizeof *cwd_file);
    cwd_n_alloc *= 2;
}

f = &cwd_file[cwd_n_used];
memset (f, '\0', sizeof *f);
f->stat.st_ino = inode;
f->filetype = type;

f->quoted = -1;
if (! cwd_some_quoted) && align_variable_outer_quotes)
{
    /* Determine if any quoted for padding purposes. */
    f->quoted = needs_quoting (name);
    if (f->quoted)
        cwd_some_quoted = 1;
}

if (command_line_arg
    || print_hyperlink
    || format_needs_stat
    /* When coloring a directory (we may know the type from
       direct.d_type), we have to stat it in order to indicate
       sticky and/or other-writable attributes. */
    || (type == directory && print_with_color
        && (is_colored (C_OTHER_WRITABLE)
            || is_colored (C_STICKY)
            || is_colored (C_STICKY_OTHER_WRITABLE)))
    /* When dereferencing symlinks, the inode and type must come from

```

```

stat, but readdir provides the inode and type of lstat. */
|| ((print_inode || format_needs_type)
    && (type == symbolic_link || type == unknown)
    && (dereference == DEREF_ALWAYS
        || color_symlink_as_referent || check_symlink_mode))
/* Command line dereferences are already taken care of by the above
   assertion that the inode number is not yet known. */
|| (print_inode && inode == NOT_AN_INODE_NUMBER)
|| (format_needs_type
    && (type == unknown || command_line_arg
        /* --indicator-style=classify (aka -F)
           requires that we stat each regular file
           to see if it's executable. */
        || (type == normal && (indicator_style == classify
            /* This is so that --color ends up
               highlighting files with these mode
               bits set even when options like -F are
               not specified. Note we do a redundant
               stat in the very unlikely case where
               C_CAP is set but not the others. */
            || (print_with_color
                && (is_colored (C_EXEC)
                    || is_colored (C_SETUID)
                    || is_colored (C_SETGID)
                    || is_colored (C_CAP)))
            )))))
}

/* Absolute name of this file. */
char *full_name;
bool do_deref;
int err;

if (name[0] == '/' || dirname[0] == 0)
    full_name = (char *) name;
else
{
    full_name = alloca (strlen (name) + strlen (dirname) + 2);
    attach (full_name, dirname, name);
}

if (print_hyperlink)
{
    f->absolute_name = canonicalize_filename_mode (full_name,
                                                CAN_MISSING);
    if (! f->absolute_name)
        file_failure (command_line_arg,
                      _("error canonicalizing %s"), full_name);
}

switch (dereference)
{
case DEREF_ALWAYS:
    err = do_stat (full_name, &f->stat);
    do_deref = true;
    break;

case DEREF_COMMAND_LINE_ARGUMENTS:
case DEREF_COMMAND_LINE_SYMLINK_TO_DIR:
    if (command_line_arg)

```

```

{
bool need_lstat;
err = do_stat (full_name, &f->stat);
do_deref = true;

if (dereference == DEREF_COMMAND_LINE_ARGUMENTS)
    break;

need_lstat = (err < 0
    ? (errno == ENOENT || errno == ELOOP)
    : ! S_ISDIR (f->stat.st_mode));
if (!need_lstat)
    break;

/* stat failed because of ENOENT || ELOOP, maybe indicating a
   non-traversable symlink. Or stat succeeded,
   FULL_NAME does not refer to a directory,
   and --dereference-command-line-symlink-to-dir is in effect.
   Fall through so that we call lstat instead. */
}

FALLTHROUGH;

default: /* DEREF_NEVER */
err = do_lstat (full_name, &f->stat);
do_deref = false;
break;
}

if (err != 0)
{
/* Failure to stat a command line argument leads to
   an exit status of 2. For other files, stat failure
   provokes an exit status of 1. */
file_failure (command_line_arg,
    _("cannot access %s"), full_name);

f->scontext = UNKNOWN_SECURITY_CONTEXT;

if (command_line_arg)
    return 0;

f->name = xstrdup (name);
cwd_n_used++;

return 0;
}

f->stat_ok = true;

/* Note has_capability() adds around 30% runtime to 'ls --color' */
if ((type == normal || S_ISREG (f->stat.st_mode))
    && print_with_color && is_colored (C_CAP))
    f->has_capability = has_capability_cache (full_name, f);

if (format == long_format || print_scontext)
{
    bool have_scontext = false;
    bool have_acl = false;
    int attr_len = getfilecon_cache (full_name, f, do_deref);
    err = (attr_len < 0);
}

```

```

if (err == 0)
{
    if (is_smack_enabled ())
        have_scontext = ! STREQ ("_", f->scontext);
    else
        have_scontext = ! STREQ ("unlabeled", f->scontext);
}
else
{
    f->scontext = UNKNOWN_SECURITY_CONTEXT;

/* When requesting security context information, don't make
   ls fail just because the file (even a command line argument)
   isn't on the right type of file system. I.e., a getfilecon
   failure isn't in the same class as a stat failure. */
if (is_ENOTSUP (errno) || errno == ENODATA)
    err = 0;
}

if (err == 0 && format == long_format)
{
    int n = file_has_acl_cache (full_name, f);
    err = (n < 0);
    have_acl = (0 < n);
}

f->acl_type = (!have_scontext && !have_acl
                ? ACL_T_NONE
                : (have_scontext && !have_acl
                   ? ACL_T_LSM_CONTEXT_ONLY
                   : ACL_T_YES));
any_has_acl |= f->acl_type != ACL_T_NONE;

if (err)
    error (0, errno, "%s", quotef (full_name));
}

if (S_ISLNK (f->stat.st_mode)
    && (format == long_format || check_symlink_mode))
{
    struct stat linkstats;

    get_link_name (full_name, f, command_line_arg);

    /* Use the slower quoting path for this entry, though
       don't update CWD_SOME_QUOTED since alignment not affected. */
    if (f->linkname && f->quoted == 0 && needs_quoting (f->linkname))
        f->quoted = -1;

    /* Avoid following symbolic links when possible, i.e., when
       they won't be traced and when no indicator is needed. */
    if (f->linkname
        && (file_type <= indicator_style || check_symlink_mode)
        && stat_for_mode (full_name, &linkstats) == 0)
    {
        f->linkok = true;
        f->linkmode = linkstats.st_mode;
    }
}

```

```

if (S_ISLNK (f->stat.st_mode))
    f->filetype = symbolic_link;
else if (S_ISDIR (f->stat.st_mode))
{
    if (command_line_arg && !immediate_dirs)
        f->filetype = arg_directory;
    else
        f->filetype = directory;
}
else
    f->filetype = normal;

blocks = STP_NBLOCKS (&f->stat);
if (format == long_format || print_block_size)
{
    char buf[LONGEST_HUMAN_READABLE + 1];
    int len = mbswidth (human_readable (blocks, buf, human_output_opts,
                                         ST_NBLOCKSIZE, output_block_size),
                         MBSWIDTH_FLAGS);
    if (block_size_width < len)
        block_size_width = len;
}

if (format == long_format)
{
    if (print_owner)
    {
        int len = format_user_width (f->stat.st_uid);
        if (owner_width < len)
            owner_width = len;
    }

    if (print_group)
    {
        int len = format_group_width (f->stat.st_gid);
        if (group_width < len)
            group_width = len;
    }

    if (print_author)
    {
        int len = format_user_width (f->stat.st_author);
        if (author_width < len)
            author_width = len;
    }
}

if (print_scontext)
{
    int len = strlen (f->scontext);
    if (scontext_width < len)
        scontext_width = len;
}

if (format == long_format)
{
    char b[INT_BUFSIZE_BOUND (uintmax_t)];
    int b_len = strlen (umaxtostr (f->stat.st_nlink, b));
    if (nlink_width < b_len)

```

```

nlink_width = b_len;

if (S_ISCHR (f->stat.st_mode) || S_ISBLK (f->stat.st_mode))
{
    char buf[INT_BUFSIZE_BOUND (uintmax_t)];
    int len = strlen (umaxtostr (major (f->stat.st_rdev), buf));
    if (major_device_number_width < len)
        major_device_number_width = len;
    len = strlen (umaxtostr (minor (f->stat.st_rdev), buf));
    if (minor_device_number_width < len)
        minor_device_number_width = len;
    len = major_device_number_width + 2 + minor_device_number_width;
    if (file_size_width < len)
        file_size_width = len;
}
else
{
    char buf[LONGEST_HUMAN_READABLE + 1];
    uintmax_t size = unsigned_file_size (f->stat.st_size);
    int len = mbswidth (human_readable (size, buf,
                                         file_human_output_opts,
                                         1, file_output_block_size),
                         MBSWIDTH_FLAGS);
    if (file_size_width < len)
        file_size_width = len;
}
}

if (print_inode)
{
    char buf[INT_BUFSIZE_BOUND (uintmax_t)];
    int len = strlen (umaxtostr (f->stat.st_ino, buf));
    if (inode_number_width < len)
        inode_number_width = len;
}

f->name = xstrdup (name);
cwd_n_used++;

return blocks;
}

/* Return true if F refers to a directory. */
static bool
is_directory (const struct fileinfo *f)
{
    return f->filetype == directory || f->filetype == arg_directory;
}

/* Return true if F refers to a (symlinked) directory. */
static bool
is_linked_directory (const struct fileinfo *f)
{
    return f->filetype == directory || f->filetype == arg_directory
        || S_ISDIR (f->linkmode);
}

/* Put the name of the file that FILENAME is a symbolic link to
into the LINKNAME field of 'f'. COMMAND_LINE_ARG indicates whether

```

```

FILENAME is a command-line argument. */

static void
get_link_name (char const *filename, struct fileinfo *f, bool command_line_arg)
{
f->linkname = areadlink_with_size (filename, f->stat.st_size);
if (f->linkname == nullptr)
    file_failure (command_line_arg, _("cannot read symbolic link %s"),
                  filename);
}

/* Return true if the last component of NAME is '.' or '..'
This is so we don't try to recurse on './././...'. */

static bool
basename_is_dot_or_dotdot (char const *name)
{
char const *base = last_component (name);
return dot_or_dotdot (base);
}

/* Remove any entries from CWD_FILE that are for directories,
and queue them to be listed as directories instead.
DIRNAME is the prefix to prepend to each dirname
to make it correct relative to ls's working dir;
if it is null, no prefix is needed and "." and ".." should not be ignored.
If COMMAND_LINE_ARG is true, this directory was mentioned at the top level,
This is desirable when processing directories recursively. */

static void
extract_dirs_from_files (char const *dirname, bool command_line_arg)
{
size_t i;
size_t j;
bool ignore_dot_and_dot_dot = (dirname != nullptr);

if (dirname && LOOP_DETECT)
{
/* Insert a marker entry first. When we dequeue this marker entry,
we'll know that DIRNAME has been processed and may be removed
from the set of active directories. */
queue_directory (nullptr, dirname, false);
}

/* Queue the directories last one first, because queueing reverses the
order. */
for (i = cwd_n_used; i-- != 0; )
{
struct fileinfo *f = sorted_file[i];

if (is_directory (f)
    && (! ignore_dot_and_dot_dot
         || ! basename_is_dot_or_dotdot (f->name)))
{
if (!dirname || f->name[0] == '/')
    queue_directory (f->name, f->linkname, command_line_arg);
else
{
    char *name = file_name_concat (dirname, f->name, nullptr);
    queue_directory (name, f->linkname, command_line_arg);
}
}
}
}

```

```

        free (name);
    }
    if (f->filetype == arg_directory)
        free_ent (f);
}
}

/* Now delete the directories from the table, compacting all the remaining
   entries. */

for (i = 0, j = 0; i < cwd_n_used; i++)
{
    struct fileinfo *f = sorted_file[i];
    sorted_file[j] = f;
    j += (f->filetype != arg_directory);
}
cwd_n_used = j;
}

/* Use strcoll to compare strings in this locale. If an error occurs,
   report an error and longjmp to failed_strcoll. */

static jmp_buf failed_strcoll;

static int
xstrcoll (char const *a, char const *b)
{
int diff;
errno = 0;
diff = strcoll (a, b);
if (errno)
{
    {
error (0, errno, _("cannot compare file names %s and %s"),
      quote_n (0, a), quote_n (1, b));
set_exit_status (false);
longjmp (failed_strcoll, 1);
    }
return diff;
}

/* Comparison routines for sorting the files. */

typedef void const *V;
typedef int (*qsortFunc)(V a, V b);

/* Used below in DEFINE_SORT_FUNCTIONS for _df_ sort function variants. */
static int
dirfirst_check (struct fileinfo const *a, struct fileinfo const *b,
                int (*cmp) (V, V))
{
int diff = is_linked_directory (b) - is_linked_directory (a);
return diff ? diff : cmp (a, b);
}

/* Define the 8 different sort function variants required for each sortkey.
KEY_NAME is a token describing the sort key, e.g., ctime, atime, size.
KEY_CMP_FUNC is a function to compare records based on that key, e.g.,
ctime_cmp, atime_cmp, size_cmp. Append KEY_NAME to the string,
'[rev_] [x] str{cmp|coll}[_df]_', to create each function name. */
#define DEFINE_SORT_FUNCTIONS(key_name, key_cmp_func) \

```

```

/* direct, non-dirfirst versions */
static int xstrcoll_##key_name (V a, V b) \
{ return key_cmp_func (a, b, xstrcoll); } \
ATTRIBUTE_PURE static int strcmp_##key_name (V a, V b) \
{ return key_cmp_func (a, b, strcmp); } \
\

/* reverse, non-dirfirst versions */
static int rev_xstrcoll_##key_name (V a, V b) \
{ return key_cmp_func (b, a, xstrcoll); } \
ATTRIBUTE_PURE static int rev_strcmp_##key_name (V a, V b) \
{ return key_cmp_func (b, a, strcmp); } \
\

/* direct, dirfirst versions */
static int xstrcoll_df_##key_name (V a, V b) \
{ return dirfirst_check (a, b, xstrcoll_##key_name); } \
ATTRIBUTE_PURE static int strcmp_df_##key_name (V a, V b) \
{ return dirfirst_check (a, b, strcmp_##key_name); } \
\

/* reverse, dirfirst versions */
static int rev_xstrcoll_df_##key_name (V a, V b) \
{ return dirfirst_check (a, b, rev_xstrcoll_##key_name); } \
ATTRIBUTE_PURE static int rev_strcmp_df_##key_name (V a, V b) \
{ return dirfirst_check (a, b, rev_strcmp_##key_name); } \
\

static int
cmp_ctime (struct fileinfo const *a, struct fileinfo const *b,
           int (*cmp) (char const *, char const *))
{
int diff = timespec_cmp (get_stat_ctime (&b->stat),
                        get_stat_ctime (&a->stat));
return diff ? diff : cmp (a->name, b->name);
}

static int
cmp_mtime (struct fileinfo const *a, struct fileinfo const *b,
           int (*cmp) (char const *, char const *))
{
int diff = timespec_cmp (get_stat_mtime (&b->stat),
                        get_stat_mtime (&a->stat));
return diff ? diff : cmp (a->name, b->name);
}

static int
cmp_atime (struct fileinfo const *a, struct fileinfo const *b,
           int (*cmp) (char const *, char const *))
{
int diff = timespec_cmp (get_stat_atime (&b->stat),
                        get_stat_atime (&a->stat));
return diff ? diff : cmp (a->name, b->name);
}

static int
cmp_btime (struct fileinfo const *a, struct fileinfo const *b,
           int (*cmp) (char const *, char const *))
{
int diff = timespec_cmp (get_stat_btime (&b->stat),
                        get_stat_btime (&a->stat));
return diff ? diff : cmp (a->name, b->name);
}

```

```

static int
off_cmp (off_t a, off_t b)
{
return (a > b) - (a < b);
}

static int
cmp_size (struct fileinfo const *a, struct fileinfo const *b,
           int (*cmp) (char const *, char const *))
{
int diff = off_cmp (b->stat.st_size, a->stat.st_size);
return diff ? diff : cmp (a->name, b->name);
}

static int
cmp_name (struct fileinfo const *a, struct fileinfo const *b,
           int (*cmp) (char const *, char const *))
{
return cmp (a->name, b->name);
}

/* Compare file extensions. Files with no extension are 'smallest'.
If extensions are the same, compare by file names instead. */

static int
cmp_extension (struct fileinfo const *a, struct fileinfo const *b,
               int (*cmp) (char const *, char const *))
{
char const *base1 = strrchr (a->name, '.');
char const *base2 = strrchr (b->name, '.');
int diff = cmp (base1 ? base1 : "", base2 ? base2 : "");
return diff ? diff : cmp (a->name, b->name);
}

/* Return the (cached) screen width,
for the NAME associated with the passed fileinfo F. */

static size_t
fileinfo_name_width (struct fileinfo const *f)
{
return f->width
    ? f->width
    : quote_name_width (f->name, filename_quoting_options, f->quoted);
}

static int
cmp_width (struct fileinfo const *a, struct fileinfo const *b,
            int (*cmp) (char const *, char const *))
{
int diff = fileinfo_name_width (a) - fileinfo_name_width (b);
return diff ? diff : cmp (a->name, b->name);
}

#define DEFINE_SORT_FUNCTIONS(name, cmp_name) \
static int name (struct fileinfo const *a, struct fileinfo const *b, \
                 int (*cmp) (char const *, char const *)) \
{ \
    return cmp (a->name, b->name); \
}

DEFINE_SORT_FUNCTIONS (ctime, cmp_ctime)
DEFINE_SORT_FUNCTIONS (mtime, cmp_mtime)
DEFINE_SORT_FUNCTIONS (atime, cmp_atime)
DEFINE_SORT_FUNCTIONS (bttime, cmp_bttime)
DEFINE_SORT_FUNCTIONS (size, cmp_size)
DEFINE_SORT_FUNCTIONS (name, cmp_name)
DEFINE_SORT_FUNCTIONS (extension, cmp_extension)

```

```

#define DEFINE_SORT_FUNCTIONS(width, cmp_width)

/* Compare file versions.
Unlike the other compare functions, cmp_version does not fail
because filevercmp and strcmp do not fail; cmp_version uses strcmp
instead of xstrcoll because filevercmp is locale-independent so
strcmp is its appropriate secondary.

All the other sort options need xstrcoll and strcmp variants,
because they all use xstrcoll (either as the primary or secondary
sort key), and xstrcoll has the ability to do a longjmp if strcoll fails for
locale reasons. */
static int
cmp_version (struct fileinfo const *a, struct fileinfo const *b)
{
int diff = filevercmp (a->name, b->name);
return diff ? diff : strcmp (a->name, b->name);
}

static int
xstrcoll_version (V a, V b)
{
return cmp_version (a, b);
}
static int
rev_xstrcoll_version (V a, V b)
{
return cmp_version (b, a);
}
static int
xstrcoll_df_version (V a, V b)
{
return dirfirst_check (a, b, xstrcoll_version);
}
static int
rev_xstrcoll_df_version (V a, V b)
{
return dirfirst_check (a, b, rev_xstrcoll_version);
}

```

/\* We have  $2^3$  different variants for each sort-key function  
(for 3 independent sort modes).

The function pointers stored in this array must be dereferenced as:

```
sort_variants[sort_key][use_strcmp][reverse][dirs_first]
```

Note that the order in which sort keys are listed in the function pointer  
array below is defined by the order of the elements in the time\_type and  
sort\_type enums! \*/

```
#define LIST_SORTFUNCTION_VARIANTS(key_name) \
{ \
    { \
        { xstrcoll_##key_name, xstrcoll_df_##key_name }, \
        { rev_xstrcoll_##key_name, rev_xstrcoll_df_##key_name }, \
    }, \
    { \
        { strcmp_##key_name, strcmp_df_##key_name }, \
        { rev_strcmp_##key_name, rev_strcmp_df_##key_name }, \
    } \
}
```

```

        }

static qsrtFunc const sort_functions[][2][2][2] =
{
    LIST_SORTFUNCTION_VARIANTS (name),
    LIST_SORTFUNCTION_VARIANTS (extension),
    LIST_SORTFUNCTION_VARIANTS (width),
    LIST_SORTFUNCTION_VARIANTS (size),

    {
    {
        { xstrcoll_version, xstrcoll_df_version },
        { rev_xstrcoll_version, rev_xstrcoll_df_version },
    },
    /* We use nullptr for the strcmp variants of version comparison
       since as explained in cmp_version definition, version comparison
       does not rely on xstrcoll, so it will never longjmp, and never
       need to try the strcmp fallback. */
    {
        { nullptr, nullptr },
        { nullptr, nullptr },
    }
    },
    /* last are time sort functions */
    LIST_SORTFUNCTION_VARIANTS (mtime),
    LIST_SORTFUNCTION_VARIANTS (ctime),
    LIST_SORTFUNCTION_VARIANTS (atime),
    LIST_SORTFUNCTION_VARIANTS (btme)
};

/* The number of sort keys is calculated as the sum of
   the number of elements in the sort_type enum (i.e., sort_numtypes)
   -2 because neither sort_time nor sort_none use entries themselves
   the number of elements in the time_type enum (i.e., time_numtypes)
This is because when sort_type==sort_time, we have up to
time_numtypes possible sort keys.

This line verifies at compile-time that the array of sort functions has been
initialized for all possible sort keys. */
static_assert (ARRAY_CARDINALITY (sort_functions)
              == sort_numtypes - 2 + time_numtypes);

/* Set up SORTED_FILE to point to the in-use entries in CWD_FILE, in order. */

static void
initialize_ordering_vector (void)
{
for (size_t i = 0; i < cwd_n_used; i++)
    sorted_file[i] = &cwd_file[i];
}

/* Cache values based on attributes global to all files. */

static void
update_current_files_info (void)
{
/* Cache screen width of name, if needed multiple times. */

```

```

if (sort_type == sort_width
    || (line_length && (format == many_per_line || format == horizontal)))
{
    size_t i;
    for (i = 0; i < cwd_n_used; i++)
    {
        struct fileinfo *f = sorted_file[i];
        f->width = fileinfo_name_width (f);
    }
}
}

/* Sort the files now in the table. */

static void
sort_files (void)
{
bool use_strcmp;

if (sorted_file_alloc < cwd_n_used + cwd_n_used / 2)
{
    free (sorted_file);
    sorted_file = xnmalloc (cwd_n_used, 3 * sizeof *sorted_file);
    sorted_file_alloc = 3 * cwd_n_used;
}

initialize_ordering_vector ();

update_current_files_info ();

if (sort_type == sort_none)
    return;

/* Try strcoll. If it fails, fall back on strcmp. We can't safely
 ignore strcoll failures, as a failing strcoll might be a
 comparison function that is not a total order, and if we ignored
 the failure this might cause qsort to dump core. */

if (! setjmp (failed_strcoll))
    use_strcmp = false; /* strcoll() succeeded */
else
{
    use_strcmp = true;
    affirm (sort_type != sort_version);
    initialize_ordering_vector ();
}

/* When sort_type == sort_time, use time_type as subindex. */
mpsort ((void const **) sorted_file, cwd_n_used,
        sort_functions[sort_type + (sort_type == sort_time ? time_type : 0)]
            [use_strcmp][sort_reverse]
            [directories_first]);
}

/* List all the files now in the table. */

static void
print_current_files (void)
{
size_t i;

```

```

switch (format)
{
    case one_per_line:
        for (i = 0; i < cwd_n_used; i++)
        {
            print_file_name_and_frills (sorted_file[i], 0);
            putchar (eolbyte);
        }
        break;

    case many_per_line:
        if (!line_length)
            print_with_separator (' ');
        else
            print_many_per_line ();
        break;

    case horizontal:
        if (!line_length)
            print_with_separator (' ');
        else
            print_horizontal ();
        break;

    case with_commas:
        print_with_separator (',');
        break;

    case long_format:
        for (i = 0; i < cwd_n_used; i++)
        {
            set_normal_color ();
            print_long_format (sorted_file[i]);
            dired_outbyte (eolbyte);
        }
        break;
    }

/* Replace the first %b with precomputed aligned month names.
Note on glibc-2.7 at least, this speeds up the whole 'ls -lU'
process by around 17%, compared to letting strftime() handle the %b. */

static size_t
align_nstrftime (char *buf, size_t size, bool recent, struct tm const *tm,
                 timezone_t tz, int ns)
{
    char const *nfmt = (use_abformat
                        ? abformat[recent][tm->tm_mon]
                        : long_time_format[recent]);
    return nstrftime (buf, size, nfmt, tm, tz, ns);
}

/* Return the expected number of columns in a long-format timestamp,
or zero if it cannot be calculated. */

static int
long_time_expected_width (void)
{

```

```

static int width = -1;

if (width < 0)
{
    time_t epoch = 0;
    struct tm tm;
    char buf[TIME_STAMP_LEN_MAXIMUM + 1];

    /* In case you're wondering if localtime_rz can fail with an input time_t
     * value of 0, let's just say it's very unlikely, but not inconceivable.
     * The TZ environment variable would have to specify a time zone that
     * is 2**31-1900 years or more ahead of UTC. This could happen only on
     * a 64-bit system that blindly accepts e.g., TZ=UTC+2000000000000000.
     * However, this is not possible with Solaris 10 or glibc-2.3.5, since
     * their implementations limit the offset to 167:59 and 24:00, resp. */
    if (localtime_rz (localtz, &epoch, &tm))
    {
        size_t len = align_nstrftime (buf, sizeof buf, false,
                                     &tm, localtz, 0);
        if (len != 0)
            width = mbsnwidth (buf, len, MBSWIDTH_FLAGS);
    }

    if (width < 0)
        width = 0;
}

return width;
}

/* Print the user or group name NAME, with numeric id ID, using a
print width of WIDTH columns. */

static void
format_user_or_group (char const *name, uintmax_t id, int width)
{
if (name)
{
    int name_width = mbswidth (name, MBSWIDTH_FLAGS);
    int width_gap = name_width < 0 ? 0 : width - name_width;
    int pad = MAX (0, width_gap);
    dired_outstring (name);

    do
        dired_outbyte (' ');
    while (pad--);
}
else
    dired_pos += printf ("%*ju ", width, id);
}

/* Print the name or id of the user with id U, using a print width of
WIDTH. */

static void
format_user (uid_t u, int width, bool stat_ok)
{
format_user_or_group (! stat_ok ? "?" :
                     (numeric_ids ? nullptr : getuser (u)), u, width);
}

```

```

/* Likewise, for groups. */

static void
format_group (gid_t g, int width, bool stat_ok)
{
format_user_or_group (! stat_ok ? "?" :
(numeric_ids ? nullptr : getgroup (g)), g, width);
}

/* Return the number of columns that format_user_or_group will print,
or -1 if unknown. */

static int
format_user_or_group_width (char const *name, uintmax_t id)
{
return (name
? mbswidth (name, MBSWIDTH_FLAGS)
: snprintf (nullptr, 0, "%ju", id));
}

/* Return the number of columns that format_user will print,
or -1 if unknown. */

static int
format_user_width (uid_t u)
{
return format_user_or_group_width (numeric_ids ? nullptr : getuser (u), u);
}

/* Likewise, for groups. */

static int
format_group_width (gid_t g)
{
return format_user_or_group_width (numeric_ids ? nullptr : getgroup (g), g);
}

/* Return a pointer to a formatted version of F->stat.st_ino,
possibly using buffer, which must be at least
INT_BUFSIZE_BOUND (uintmax_t) bytes. */
static char *
format_inode (char buf[INT_BUFSIZE_BOUND (uintmax_t)],
const struct fileinfo *f)
{
return (f->stat_ok && f->stat.st_ino != NOT_AN_INODE_NUMBER
? umaxtostr (f->stat.st_ino, buf)
: (char *) "?");
}

/* Print information about F in long format. */
static void
print_long_format (const struct fileinfo *f)
{
char modebuf[12];
char buf
[LONGEST_HUMAN_READABLE + 1          /* inode */
+ LONGEST_HUMAN_READABLE + 1        /* size in blocks */
+ sizeof (modebuf) - 1 + 1         /* mode string */
+ INT_BUFSIZE_BOUND (uintmax_t)    /* st_nlink */

```

```

+ LONGEST_HUMAN_READABLE + 2 /* major device number */
+ LONGEST_HUMAN_READABLE + 1 /* minor device number */
+ TIME_STAMP_LEN_MAXIMUM + 1 /* max length of time/date */
];
size_t s;
char *p;
struct timespec when_timespec;
struct tm when_local;
bool btime_ok = true;

/* Compute the mode string, except remove the trailing space if no
   file in this directory has an ACL or security context. */
if (f->stat_ok)
    filemodestring (&f->stat, modebuf);
else
{
    modebuf[0] = filetype_letter[f->filetype];
    memset (modebuf + 1, '?', 10);
    modebuf[11] = '\0';
}
if (! any_has_acl)
    modebuf[10] = '\0';
else if (f->acl_type == ACL_T_LSM_CONTEXT_ONLY)
    modebuf[10] = ':';
else if (f->acl_type == ACL_T_YES)
    modebuf[10] = '+';

switch (time_type)
{
case time_ctime:
when_timespec = get_stat_ctime (&f->stat);
break;
case time_mtime:
when_timespec = get_stat_mtime (&f->stat);
break;
case time_atime:
when_timespec = get_stat_atime (&f->stat);
break;
case time_btime:
when_timespec = get_stat_btime (&f->stat);
if (when_timespec.tv_sec == -1 && when_timespec.tv_nsec == -1)
    btime_ok = false;
break;
default:
unreachable ();
}

p = buf;

if (print_inode)
{
    char hbuf[INT_BUFSIZE_BOUND (uintmax_t)];
    p += sprintf (p, "%*s ", inode_number_width, format_inode (hbuf, f));
}

if (print_block_size)
{
    char hbuf[LONGEST_HUMAN_READABLE + 1];
    char const *blocks =
        (! f->stat_ok

```

```

? "?"
: human_readable (STP_NBLOCKS (&f->stat), hbuf, human_output_opts,
                  ST_NBLOCKSIZE, output_block_size));
int blocks_width = mbswidth (blocks, MBSWIDTH_FLAGS);
for (int pad = blocks_width < 0 ? 0 : block_size_width - blocks_width;
     0 < pad; pad--)
    *p++ = ' ';
while ((*p++ = *blocks++) != '\0')
    continue;
p[-1] = ' ';
}

/* The last byte of the mode string is the POSIX
   "optional alternate access method flag". */
{
    char hbuf[INT_BUFSIZE_BOUND (uintmax_t)];
    p += sprintf (p, "%s %*s ", modebuf, nlink_width,
                  ! f->stat_ok ? "?" : umaxtostr (f->stat.st_nlink, hbuf));
}
dired_indent ();

if (print_owner || print_group || print_author || print_scontext)
{
    dired_outbuf (buf, p - buf);

    if (print_owner)
        format_user (f->stat.st_uid, owner_width, f->stat_ok);

    if (print_group)
        format_group (f->stat.st_gid, group_width, f->stat_ok);

    if (print_author)
        format_user (f->stat.st_author, author_width, f->stat_ok);

    if (print_scontext)
        format_user_or_group (f->scontext, 0, scontext_width);

    p = buf;
}

if (f->stat_ok
    && (S_ISCHR (f->stat.st_mode) || S_ISBLK (f->stat.st_mode)))
{
    char majorbuf[INT_BUFSIZE_BOUND (uintmax_t)];
    char minorbuf[INT_BUFSIZE_BOUND (uintmax_t)];
    int blanks_width = (file_size_width
                        - (major_device_number_width + 2
                           + minor_device_number_width));
    p += sprintf (p, "%*s, %*s ",
                  major_device_number_width + MAX (0, blanks_width),
                  umaxtostr (major (f->stat.st_rdev), majorbuf),
                  minor_device_number_width,
                  umaxtostr (minor (f->stat.st_rdev), minorbuf));
}
else
{
    char hbuf[LONGEST_HUMAN_READABLE + 1];
    char const *size =
        (! f->stat_ok

```

```

? "?"
: human_readable (unsigned_file_size (f->stat.st_size),
                  hbuf, file_human_output_opts, 1,
                  file_output_block_size));
int size_width = mbswidth (size, MBSWIDTH_FLAGS);
for (int pad = size_width < 0 ? 0 : file_size_width - size_width;
     0 < pad; pad--)
    *p++ = ' ';
while ((*p++ = *size++) == '\0')
    continue;
p[-1] = '\0';
}

s = 0;
*p = '\0';

if (f->stat_ok && btime_ok
    && localtime_rz (localtz, &when_timespec.tv_sec, &when_local))
{
    struct timespec six_months_ago;
    bool recent;

    /* If the file appears to be in the future, update the current
       time, in case the file happens to have been modified since
       the last time we checked the clock. */
    if (timespec_cmp (current_time, when_timespec) < 0)
        gettimeofday (&current_time);

    /* Consider a time to be recent if it is within the past six months.
       A Gregorian year has 365.2425 * 24 * 60 * 60 == 31556952 seconds
       on the average. Write this value as an integer constant to
       avoid floating point hassles. */
    six_months_ago.tv_sec = current_time.tv_sec - 31556952 / 2;
    six_months_ago.tv_nsec = current_time.tv_nsec;

    recent = (timespec_cmp (six_months_ago, when_timespec) < 0
              && timespec_cmp (when_timespec, current_time) < 0);

    /* We assume here that all time zones are offset from UTC by a
       whole number of seconds. */
    s = align_nstrftime (p, TIME_STAMP_LEN_MAXIMUM + 1, recent,
                         &when_local, localtz, when_timespec.tv_nsec);
}

if (s || !*p)
{
    p += s;
    *p++ = ' ';
}
else
{
    /* The time cannot be converted using the desired format, so
       print it as a huge integer number of seconds. */
    char hbuf[INT_BUFSIZE_BOUND (intmax_t)];
    p += sprintf (p, "%*s", long_time_expected_width (),
                  (! f->stat_ok || ! btime_ok
                   ? "?" :
                   timetostr (when_timespec.tv_sec, hbuf)));
    /* FIXME: (maybe) We discarded when_timespec.tv_nsec. */
}

```

```

dired_outbuf (buf, p - buf);
size_t w = print_name_with_quoting (f, false, &dired_obstack, p - buf);

if (f->filetype == symbolic_link)
{
    if (f->linkname)
    {
        dired_outstring (" -> ");
        print_name_with_quoting (f, true, nullptr, (p - buf) + w + 4);
        if (indicator_style != none)
            print_type_indicator (true, f->linkmode, unknown);
    }
}
else if (indicator_style != none)
    print_type_indicator (f->stat_ok, f->stat.st_mode, f->filetype);
}

/* Write to *BUF a quoted representation of the file name NAME, if non-null,
using OPTIONS to control quoting. *BUF is set to NAME if no quoting
is required. *BUF is allocated if more space required (and the original
*BUF is not deallocated).
Store the number of screen columns occupied by NAME's quoted
representation into WIDTH, if non-null.
Store into PAD whether an initial space is needed for padding.
Return the number of bytes in *BUF. */


```

```

static size_t
quote_name_buf (char **inbuf, size_t bufsize, char *name,
                struct quoting_options const *options,
                int needs_general_quoting, size_t *width, bool *pad)
{
    char *buf = *inbuf;
    size_t displayed_width IF_LINT (= 0);
    size_t len = 0;
    bool quoted;

    enum quoting_style qs = get_quoting_style (options);
    bool needs_further_quoting = qmark_funny_chars
        && (qs == shell_quoting_style
            || qs == shell_always_quoting_style
            || qs == literal_quoting_style);

    if (needs_general_quoting != 0)
    {
        len = quotearg_buffer (buf, bufsize, name, -1, options);
        if (bufsize <= len)
        {
            buf = xmalloc (len + 1);
            quotearg_buffer (buf, len + 1, name, -1, options);
        }

        quoted = (*name != *buf) || strlen (name) != len;
    }
    else if (needs_further_quoting)
    {
        len = strlen (name);
        if (bufsize <= len)
            buf = xmalloc (len + 1);
        memcpy (buf, name, len + 1);
    }
}
```

```

        quoted = false;
    }
else
{
    len = strlen (name);
    buf = name;
    quoted = false;
}

if (needs_further_quoting)
{
    if (MB_CUR_MAX > 1)
    {
        char const *p = buf;
        char const *plimit = buf + len;
        char *q = buf;
        displayed_width = 0;

        while (p < plimit)
            switch (*p)
            {
                case ' ': case '!': case "'": case '#': case '%':
                case '&': case '\\': case '(': case ')': case '*':
                case '+': case ',': case '-': case '.': case '/':
                case '0': case '1': case '2': case '3': case '4':
                case '5': case '6': case '7': case '8': case '9':
                case ':': case ';': case '<': case '=': case '>':
                case '?':
                case 'A': case 'B': case 'C': case 'D': case 'E':
                case 'F': case 'G': case 'H': case 'I': case 'J':
                case 'K': case 'L': case 'M': case 'N': case 'O':
                case 'P': case 'Q': case 'R': case 'S': case 'T':
                case 'U': case 'V': case 'W': case 'X': case 'Y':
                case 'Z':
                case '[': case '\\': case ']': case '^': case '_':
                case 'a': case 'b': case 'c': case 'd': case 'e':
                case 'f': case 'g': case 'h': case 'i': case 'j':
                case 'k': case 'l': case 'm': case 'n': case 'o':
                case 'p': case 'q': case 'r': case 's': case 't':
                case 'u': case 'v': case 'w': case 'x': case 'y':
                case 'z': case '{': case '}': case '|': case '~':
                /* These characters are printable ASCII characters. */
                *q++ = *p++;
                displayed_width += 1;
                break;
            default:
                /* If we have a multibyte sequence, copy it until we
                   reach its end, replacing each non-printable multibyte
                   character with a single question mark. */
            {
                mbstate_t mbstate; mbszero (&mbstate);
                do
                {
                    char32_t wc;
                    size_t bytes;
                    int w;

                    bytes = mbrtoc32 (&wc, p, plimit - p, &mbstate);

```

```

        if (bytes == (size_t) -1)
        {
            /* An invalid multibyte sequence was
               encountered. Skip one input byte, and
               put a question mark. */
            p++;
            *q++ = '?';
            displayed_width += 1;
            break;
        }

        if (bytes == (size_t) -2)
        {
            /* An incomplete multibyte character
               at the end. Replace it entirely with
               a question mark. */
            p = plimit;
            *q++ = '?';
            displayed_width += 1;
            break;
        }

        if (bytes == 0)
        /* A null wide character was encountered. */
        bytes = 1;

        w = c32width (wc);
        if (w >= 0)
        {
            /* A printable multibyte character.
               Keep it. */
            for (; bytes > 0; --bytes)
                *q++ = *p++;
            displayed_width += w;
        }
        else
        {
            /* An nonprintable multibyte character.
               Replace it entirely with a question
               mark. */
            p += bytes;
            *q++ = '?';
            displayed_width += 1;
        }
    }
    while (! mbsinit (&mbstate));
}
break;
}

/* The buffer may have shrunk. */
len = q - buf;
}
else
{
    char *p = buf;
    char const *plimit = buf + len;

    while (p < plimit)
    {

```

```

        if (!isprint (to_uchar (*p)))
            *p = '?';
        p++;
    }
    displayed_width = len;
}
else if (width != nullptr)
{
    if (MB_CUR_MAX > 1)
    {
        displayed_width = mbsnwidth (buf, len, MBSWIDTH_FLAGS);
        displayed_width = MAX (0, displayed_width);
    }
else
{
    char const *p = buf;
    char const *plimit = buf + len;

    displayed_width = 0;
    while (p < plimit)
    {
        if (isprint (to_uchar (*p)))
            displayed_width++;
        p++;
    }
}
/* Set padding to better align quoted items,
   and also give a visual indication that quotes are
   not actually part of the name. */
*pad = (align_variable_outer_quotes && cwd_some_quoted && !quoted);

if (width != nullptr)
    *width = displayed_width;

*inbuf = buf;

return len;
}

static size_t
quote_name_width (char const *name, struct quoting_options const *options,
                  int needs_general_quoting)
{
    char smallbuf[BUFSIZ];
    char *buf = smallbuf;
    size_t width;
    bool pad;

    quote_name_buf (&buf, sizeof smallbuf, (char *) name, options,
                   needs_general_quoting, &width, &pad);

    if (buf != smallbuf && buf != name)
        free (buf);

    width += pad;

    return width;
}

```

```

}

/* %XX escape any input out of range as defined in RFC3986,
and also if PATH, convert all path separators to '/'. */
static char *
file_escape (char const *str, bool path)
{
char *esc = xnmalloc (3, strlen (str) + 1);
char *p = esc;
while (*str)
{
if (path && ISSLASH (*str))
{
*p++ = '/';
str++;
}
else if (RFC3986[to_uchar (*str)])
*p++ = *str++;
else
p += sprintf (p, "%%%02x", to_uchar (*str++));
}
*p = '\0';
return esc;
}

static size_t
quote_name (char const *name, struct quoting_options const *options,
int needs_general_quoting, const struct bin_str *color,
bool allow_pad, struct obstack *stack, char const *absolute_name)
{
char smallbuf[BUFSIZ];
char *buf = smallbuf;
size_t len;
bool pad;

len = quote_name_buf (&buf, sizeof smallbuf, (char *) name, options,
needs_general_quoting, nullptr, &pad);

if (pad && allow_pad)
dired_outbyte (' ');

if (color)
print_color_indicator (color);

/* If we're padding, then don't include the outer quotes in
the --hyperlink, to improve the alignment of those links. */
bool skip_quotes = false;

if (absolute_name)
{
if (align_variable_outer_quotes && cwd_some_quoted && ! pad)
{
skip_quotes = true;
putchar (*buf);
}
char *h = file_escape (hostname, /* path= */ false);
char *n = file_escape (absolute_name, /* path= */ true);
/* TODO: It would be good to be able to define parameters
to give hints to the terminal as how best to render the URI.
For example since ls is outputting a dense block of URIs

```

```

it would be best to not underline by default, and only
do so upon hover etc. */
printf ("\033]8;;file://%s%s%s\a", h, *n == '/' ? "" : "/", n);
free (h);
free (n);
}

if (stack)
    push_current_dired_pos (stack);

fwrite (buf + skip_quotes, 1, len - (skip_quotes * 2), stdout);

dired_pos += len;

if (stack)
    push_current_dired_pos (stack);

if (absolute_name)
{
    fputs ("\033]8;\\a", stdout);
    if (skip_quotes)
        putchar (*(buf + len - 1));
}

if (buf != smallbuf && buf != name)
    free (buf);

return len + pad;
}

static size_t
print_name_with_quoting (const struct fileinfo *f,
                        bool symlink_target,
                        struct obstack *stack,
                        size_t start_col)
{
char const *name = symlink_target ? f->linkname : f->name;

const struct bin_str *color
    = print_with_color ? get_color_indicator (f, symlink_target) : nullptr;

bool used_color_this_time = (print_with_color
    && (color || is_colored (C_NORM)));

size_t len = quote_name (name, filename_quoting_options, f->quoted,
                        color, !symlink_target, stack, f->absolute_name);

process_signals ();
if (used_color_this_time)
{
    prep_non_filename_text ();

/* We use the byte length rather than display width here as
   an optimization to avoid accurately calculating the width,
   because we only output the clear to EOL sequence if the name
   _might_ wrap to the next line. This may output a sequence
   unnecessarily in multi-byte locales for example,
   but in that case it's inconsequential to the output. */
if (line_length
    && (start_col / line_length != (start_col + len - 1) / line_length))

```

```

        put_indicator (&color_indicator[C_CLR_TO_EOL]);
    }

return len;
}

static void
prep_non_filename_text (void)
{
if (color_indicator[C_END].string != nullptr)
    put_indicator (&color_indicator[C_END]);
else
{
    {
    put_indicator (&color_indicator[C_LEFT]);
    put_indicator (&color_indicator[C_RESET]);
    put_indicator (&color_indicator[C_RIGHT]);
    }
}
}

/* Print the file name of 'f' with appropriate quoting.
Also print file size, inode number, and filetype indicator character,
as requested by switches. */

static size_t
print_file_name_and_frills (const struct fileinfo *f, size_t start_col)
{
char buf[MAX (LONGEST_HUMAN_READABLE + 1, INT_BUFSIZE_BOUND (uintmax_t))];

set_normal_color ();

if (print_inode)
    printf ("%*s ", format == with_commas ? 0 : inode_number_width,
            format_inode (buf, f));

if (print_block_size)
    printf ("%*s ", format == with_commas ? 0 : block_size_width,
            ! f->stat_ok ? "?" :
            : human_readable (STP_NBLOCKS (&f->stat), buf, human_output_opts,
                            ST_NBLOCKSIZE, output_block_size));

if (print_scontext)
    printf ("%*s ", format == with_commas ? 0 : scontext_width, f->scontext);

size_t width = print_name_with_quoting (f, false, nullptr, start_col);

if (indicator_style != none)
    width += print_type_indicator (f->stat_ok, f->stat.st_mode, f->filetype);

return width;
}

/* Given these arguments describing a file, return the single-byte
type indicator, or 0. */
static char
get_type_indicator (bool stat_ok, mode_t mode, enum filetype type)
{
char c;

if (stat_ok ? S_ISREG (mode) : type == normal)
{

```

```

    if (stat_ok && indicator_style == classify && (mode & S_IXUGO))
        c = '*';
    else
        c = 0;
    }
else
{
    if (stat_ok ? S_ISDIR (mode) : type == directory || type == arg_directory)
        c = '/';
    else if (indicator_style == slash)
        c = 0;
    else if (stat_ok ? S_ISLNK (mode) : type == symbolic_link)
        c = '@';
    else if (stat_ok ? S_ISFIFO (mode) : type == fifo)
        c = '|';
    else if (stat_ok ? S_ISSOCK (mode) : type == sock)
        c = '=';
    else if (stat_ok && S_ISDOOR (mode))
        c = '>';
    else
        c = 0;
}
return c;
}

static bool
print_type_indicator (bool stat_ok, mode_t mode, enum filetype type)
{
char c = get_type_indicator (stat_ok, mode, type);
if (c)
    dired_outbyte (c);
return !c;
}

/* Returns if color sequence was printed. */
static bool
print_color_indicator (const struct bin_str *ind)
{
if (ind)
{
    /* Need to reset so not dealing with attribute combinations */
    if (is_colored (C_NORM))
        restore_default_color ();
    put_indicator (&color_indicator[C_LEFT]);
    put_indicator (ind);
    put_indicator (&color_indicator[C_RIGHT]);
}

return ind != nullptr;
}

/* Returns color indicator or nullptr if none. */
ATTRIBUTE PURE
static const struct bin_str*
get_color_indicator (const struct fileinfo *f, bool symlink_target)
{
enum indicator_no type;
struct color_ext_type *ext; /* Color extension */
size_t len; /* Length of name */

```

```

char const *name;
mode_t mode;
int linkok;
if (symlink_target)
{
    name = f->linkname;
    mode = f->linkmode;
    linkok = f->linkok ? 0 : -1;
}
else
{
    name = f->name;
    mode = file_or_link_mode (f);
    linkok = f->linkok;
}

/* Is this a nonexistent file? If so, linkok == -1. */

if (linkok == -1 && is_colored (C_MISSING))
    type = C_MISSING;
else if (!f->stat_ok)
{
    static enum indicator_no filetype_indicator[] = FILETYPE_INDICATORS;
    type = filetype_indicator[f->filetype];
}
else
{
    if (S_ISREG (mode))
    {
        type = C_FILE;

        if ((mode & S_ISUID) != 0 && is_colored (C_SETUID))
            type = C_SETUID;
        else if ((mode & S_ISGID) != 0 && is_colored (C_SETGID))
            type = C_SETGID;
        else if (is_colored (C_CAP) && f->has_capability)
            type = C_CAP;
        else if ((mode & S_IXUGO) != 0 && is_colored (C_EXEC))
            type = C_EXEC;
        else if ((1 < f->stat.st_nlink) && is_colored (C_MULTIHARDLINK))
            type = C_MULTIHARDLINK;
    }
    else if (S_ISDIR (mode))
    {
        type = C_DIR;

        if ((mode & S_ISVTX) && (mode & S_IWOTH)
            && is_colored (C_STICKY_OTHER_WRITABLE))
            type = C_STICKY_OTHER_WRITABLE;
        else if ((mode & S_IWOTH) != 0 && is_colored (C_OTHER_WRITABLE))
            type = C_OTHER_WRITABLE;
        else if ((mode & S_ISVTX) != 0 && is_colored (C_STICKY))
            type = C_STICKY;
    }
    else if (S_ISLNK (mode))
        type = C_LINK;
    else if (S_ISFIFO (mode))
        type = C_FIFO;
    else if (S_ISSOCK (mode))
        type = C_SOCK;
}

```

```

else if (S_ISBLK (mode))
    type = C_BLK;
else if (S_ISCHR (mode))
    type = C_CHR;
else if (S_ISDOOR (mode))
    type = C_DOOR;
else
{
    /* Classify a file of some other type as C_ORPHAN. */
    type = C_ORPHAN;
}
}

/* Check the file's suffix only if still classified as C_FILE. */
ext = nullptr;
if (type == C_FILE)
{
    /* Test if NAME has a recognized suffix. */

    len = strlen (name);
    name += len;           /* Pointer to final \0. */
    for (ext = color_ext_list; ext != nullptr; ext = ext->next)
    {
        if (ext->ext.len <= len)
        {
            if (ext->exact_match)
            {
                if (STREQ_LEN (name - ext->ext.len, ext->ext.string,
                               ext->ext.len))
                    break;
            }
            else
            {
                if (c_strncasecmp (name - ext->ext.len, ext->ext.string,
                                   ext->ext.len) == 0)
                    break;
            }
        }
    }
}

/* Adjust the color for orphaned symlinks. */
if (type == C_LINK && !linkok)
{
    if (color_symlink_as_referent || is_colored (C_ORPHAN))
        type = C_ORPHAN;
}

const struct bin_str *const s
= ext ? &(ext->seq) : &color_indicator[type];

return s->string ? s : nullptr;
}

/* Output a color indicator (which may contain nulls). */
static void
put_indicator (const struct bin_str *ind)
{
if (! used_color)
{

```

```

used_color = true;

/* If the standard output is a controlling terminal, watch out
   for signals, so that the colors can be restored to the
   default state if "ls" is suspended or interrupted. */

if (0 <= tcgetpgrp (STDOUT_FILENO))
    signal_init ();

prep_non_filename_text ();
}

fwrite (ind->string, ind->len, 1, stdout);
}

static size_t
length_of_file_name_and_frills (const struct fileinfo *f)
{
size_t len = 0;
char buf[MAX (LONGEST_HUMAN_READABLE + 1, INT_BUFSIZE_BOUND (uintmax_t))];

if (print_inode)
    len += 1 + (format == with_commas
                 ? strlen (umaxtostr (f->stat.st_ino, buf))
                 : inode_number_width);

if (print_block_size)
    len += 1 + (format == with_commas
                 ? strlen (! f->stat_ok ? "?"
                           : human_readable (STP_NBLOCKS (&f->stat), buf,
                                             human_output_opts, ST_NBLOCKSIZE,
                                             output_block_size))
                 : block_size_width);

if (print_scontext)
    len += 1 + (format == with_commas ? strlen (f->scontext) : scontext_width);

len += fileinfo_name_width (f);

if (indicator_style != none)
{
    char c = get_type_indicator (f->stat_ok, f->stat.st_mode, f->filetype);
    len += (c != 0);
}

return len;
}

static void
print_many_per_line (void)
{
size_t row;           /* Current row. */
size_t cols = calculate_columns (true);
struct column_info const *line_fmt = &column_info[cols - 1];

/* Calculate the number of rows that will be in each column except possibly
   for a short column on the right. */
size_t rows = cwd_n_used / cols + (cwd_n_used % cols != 0);

for (row = 0; row < rows; row++)

```

```

{
size_t col = 0;
size_t filesno = row;
size_t pos = 0;

/* Print the next row. */
while (true)
{
    struct fileinfo const *f = sorted_file[filesno];
    size_t name_length = length_of_file_name_and_frills (f);
    size_t max_name_length = line_fmt->col_arr[col++];
    print_file_name_and_frills (f, pos);

    filesno += rows;
    if (filesno >= cwd_n_used)
        break;

    indent (pos + name_length, pos + max_name_length);
    pos += max_name_length;
}
putchar (eolbyte);
}

static void
print_horizontal (void)
{
size_t filesno;
size_t pos = 0;
size_t cols = calculate_columns (false);
struct column_info const *line_fmt = &column_info[cols - 1];
struct fileinfo const *f = sorted_file[0];
size_t name_length = length_of_file_name_and_frills (f);
size_t max_name_length = line_fmt->col_arr[0];

/* Print first entry. */
print_file_name_and_frills (f, 0);

/* Now the rest. */
for (filesno = 1; filesno < cwd_n_used; ++filesno)
{
    size_t col = filesno % cols;

    if (col == 0)
    {
        putchar (eolbyte);
        pos = 0;
    }
    else
    {
        indent (pos + name_length, pos + max_name_length);
        pos += max_name_length;
    }

    f = sorted_file[filesno];
    print_file_name_and_frills (f, pos);

    name_length = length_of_file_name_and_frills (f);
    max_name_length = line_fmt->col_arr[col];
}
}

```

```

putchar (eolbyte);
}

/* Output name + SEP + ' '. */

static void
print_with_separator (char sep)
{
size_t filesno;
size_t pos = 0;

for (filesno = 0; filesno < cwd_n_used; filesno++)
{
    struct fileinfo const *f = sorted_file[filesno];
    size_t len = line_length ? length_of_file_name_and_frills (f) : 0;

    if (filesno != 0)
    {
        char separator;

        if (!line_length
            || ((pos + len + 2 < line_length)
                && (pos <= SIZE_MAX - len - 2)))
        {
            pos += 2;
            separator = ' ';
        }
        else
        {
            pos = 0;
            separator = eolbyte;
        }

        putchar (sep);
        putchar (separator);
    }

    print_file_name_and_frills (f, pos);
    pos += len;
}
putchar (eolbyte);
}

/* Assuming cursor is at position FROM, indent up to position TO.
Use a TAB character instead of two or more spaces whenever possible. */

static void
indent (size_t from, size_t to)
{
while (from < to)
{
    if (tabsize != 0 && to / tabsize > (from + 1) / tabsize)
    {
        putchar ('\t');
        from += tabsize - from % tabsize;
    }
else
    {
        putchar (' ');
        from++;
    }
}

```

```

        }
    }
}

/* Put DIRNAME/NAME into DEST, handling '.' and '/' properly. */
/* FIXME: maybe remove this function someday. See about using a
non-malloc'ing version of file_name_concat. */

static void
attach (char *dest, char const *dirname, char const *name)
{
char const *dirnameep = dirname;

/* Copy dirname if it is not ". ". */
if (dirname[0] != '.' || dirname[1] != 0)
{
    while (*dirnameep)
        *dest++ = *dirnameep++;
    /* Add '/' if 'dirname' doesn't already end with it. */
    if (dirnameep > dirname && dirnameep[-1] != '/')
        *dest++ = '/';
}
while (*name)
    *dest++ = *name++;
*dest = 0;
}

/* Allocate enough column info suitable for the current number of
files and display columns, and initialize the info to represent the
narrowest possible columns. */

static void
init_column_info (size_t max_cols)
{
size_t i;

/* Currently allocated columns in column_info. */
static size_t column_info_alloc;

if (column_info_alloc < max_cols)
{
    size_t new_column_info_alloc;
    size_t *p;

    if (!max_idx || max_cols < max_idx / 2)
    {
        /* The number of columns is far less than the display width
           allows. Grow the allocation, but only so that it's
           double the current requirements. If the display is
           extremely wide, this avoids allocating a lot of memory
           that is never needed. */
        column_info = xnrealloc (column_info, max_cols,
                               2 * sizeof *column_info);
        new_column_info_alloc = 2 * max_cols;
    }
else
    {
        column_info = xnrealloc (column_info, max_idx, sizeof *column_info);
        new_column_info_alloc = max_idx;
    }
}

```

```

/* Allocate the new size_t objects by computing the triangle
   formula n * (n + 1) / 2, except that we don't need to
   allocate the part of the triangle that we've already
   allocated. Check for address arithmetic overflow. */
{
    size_t column_info_growth = new_column_info_alloc - column_info_alloc;
    size_t s = column_info_alloc + 1 + new_column_info_alloc;
    size_t t = s * column_info_growth;
    if (s < new_column_info_alloc || t / column_info_growth != s)
        xalloc_die();
    p = xnmalloc (t / 2, sizeof *p);
}

/* Grow the triangle by parceling out the cells just allocated. */
for (i = column_info_alloc; i < new_column_info_alloc; i++)
{
    column_info[i].col_arr = p;
    p += i + 1;
}

column_info_alloc = new_column_info_alloc;
}

for (i = 0; i < max_cols; ++i)
{
    size_t j;

    column_info[i].valid_len = true;
    column_info[i].line_len = (i + 1) * MIN_COLUMN_WIDTH;
    for (j = 0; j <= i; ++j)
        column_info[i].col_arr[j] = MIN_COLUMN_WIDTH;
}
}

/* Calculate the number of columns needed to represent the current set
   of files in the current display width. */

static size_t
calculate_columns (bool by_columns)
{
    size_t filesno;           /* Index into cwd_file. */
    size_t cols;              /* Number of files across. */

    /* Normally the maximum number of columns is determined by the
       screen width. But if few files are available this might limit it
       as well. */
    size_t max_cols = 0 < max_idx && max_idx < cwd_n_used ? max_idx : cwd_n_used;

    init_column_info (max_cols);

    /* Compute the maximum number of possible columns. */
    for (filesno = 0; filesno < cwd_n_used; ++filesno)
    {
        struct fileinfo const *f = sorted_file[filesno];
        size_t name_length = length_of_file_name_and_frills (f);

        for (size_t i = 0; i < max_cols; ++i)
        {
            if (column_info[i].valid_len)

```

```

    {
        size_t idx = (by_columns
            ? filesno / ((cwd_n_used + i) / (i + 1))
            : filesno % (i + 1));
        size_t real_length = name_length + (idx == i ? 0 : 2);

        if (column_info[i].col_arr[idx] < real_length)
        {
            column_info[i].line_len += (real_length
                - column_info[i].col_arr[idx]);
            column_info[i].col_arr[idx] = real_length;
            column_info[i].valid_len = (column_info[i].line_len
                < line_length);
        }
    }
}

/* Find maximum allowed columns. */
for (cols = max_cols; 1 < cols; --cols)
{
    if (column_info[cols - 1].valid_len)
        break;
}

return cols;
}

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    printf (_("Usage: %s [OPTION]... [FILE]...\n"), program_name);
    fputs (_("\
List information about the FILEs (the current directory by default).\n\
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.\n\
"), stdout);

    emit_mandatory_arg_note ();

    fputs (_("\
-a, --all      do not ignore entries starting with .\n\
-A, --almost-all   do not list implied . and ..\n\
--author      with -l, print the author of each file\n\
-b, --escape     print C-style escapes for nongraphic characters\n\
"), stdout);
    fputs (_("\
--block-size=SIZE   with -l, scale sizes by SIZE when printing them;\n\
                     e.g., '--block-size=M'; see SIZE format below\n\
\n\
"), stdout);
    fputs (_("\
-B, --ignore-backups  do not list implied entries ending with ~\n\
"), stdout);
    fputs (_("\
-c           with -lt: sort by, and show, ctime (time of last\n\
                     change of file status information);\n\
\n\
"), stdout);
}
}

```

```

with -l: show ctime and sort by name;\n\
otherwise: sort by ctime, newest first\n\
\n\
"), stdout);
    fputs (_("\
-C           list entries by columns\n\
    --color[=WHEN]      color the output WHEN; more info below\n\
-d, --directory      list directories themselves, not their contents\n\
-D, --dired          generate output designed for Emacs' dired mode\n\
"), stdout);
    fputs (_("\
-f           do not sort, enable -aU, disable -ls --color\n\
-F, --classify[=WHEN]  append indicator (one of */=>@!) to entries WHEN\n\
    --file-type        likewise, except do not append '*'\n\
"), stdout);
    fputs (_("\
--format=WORD       across -x, commas -m, horizontal -x, long -l,\n\
                    single-column -1, verbose -l, vertical -C\n\
\n\
"), stdout);
    fputs (_("\
--full-time        like -l --time-style=full-iso\n\
"), stdout);
    fputs (_("\
-g           like -l, but do not list owner\n\
"), stdout);
    fputs (_("\
--group-directories-first\n\
                    group directories before files;\n\
                    can be augmented with a --sort option, but any\n\
                    use of --sort=none (-U) disables grouping\n\
\n\
"), stdout);
    fputs (_("\
-G, --no-group      in a long listing, don't print group names\n\
"), stdout);
    fputs (_("\
-h, --human-readable   with -l and -s, print sizes like 1K 234M 2G etc.\n\
    --si              likewise, but use powers of 1000 not 1024\n\
"), stdout);
    fputs (_("\
-H, --dereference-command-line\n\
                    follow symbolic links listed on the command line\n\
"), stdout);
    fputs (_("\
--dereference-command-line-symlink-to-dir\n\
                    follow each command line symbolic link\n\
                    that points to a directory\n\
\n\
"), stdout);
    fputs (_("\
--hide=PATTERN      do not list implied entries matching shell PATTERN\n\
\n\
                    (overridden by -a or -A)\n\
\n\
"), stdout);
    fputs (_("\
--hyperlink[=WHEN]   hyperlink file names WHEN\n\
"), stdout);
    fputs (_("\

```

```

--indicator-style=WORD\n\
    append indicator with style WORD to entry names:\n\
        none (default), slash (-p),\n\
        file-type (--file-type), classify (-F)\n\
\n\
"), stdout);
    fputs (_("\
-i, --inode      print the index number of each file\n\
-l, --ignore=PATTERN  do not list implied entries matching shell PATTERN\n\
\n\
"), stdout);
    fputs (_("\
-k, --kibibytes   default to 1024-byte blocks for file system usage;\n\
\n\
used only with -s and per directory totals\n\
\n\
"), stdout);
    fputs (_("\
-l           use a long listing format\n\
"), stdout);
    fputs (_("\
-L, --dereference when showing file information for a symbolic\n\
link, show information for the file the link\n\
references rather than for the link itself\n\
\n\
"), stdout);
    fputs (_("\
-m           fill width with a comma separated list of entries\n\
\n\
"), stdout);
    fputs (_("\
-n, --numeric-uid-gid like -l, but list numeric user and group IDs\n\
-N, --literal     print entry names without quoting\n\
-o             like -l, but do not list group information\n\
-p, --indicator-style=slash\n\
    append / indicator to directories\n\
\n\
"), stdout);
    fputs (_("\
-q, --hide-control-chars print ? instead of nongraphic characters\n\
"), stdout);
    fputs (_("\
--show-control-chars show nongraphic characters as-is (the default,\n\
unless program is 'ls' and output is a terminal)\n\
\n\
"), stdout);
    fputs (_("\
-Q, --quote-name    enclose entry names in double quotes\n\
"), stdout);
    fputs (_("\
--quoting-style=WORD use quoting style WORD for entry names:\n\
    literal, locale, shell, shell-always,\n\
    shell-escape, shell-escape-always, c, escape\n\
    (overrides QUOTING_STYLE environment variable)\n\
\n\
"), stdout);
    fputs (_("\
-r, --reverse      reverse order while sorting\n\
-R, --recursive    list subdirectories recursively\n\
-s, --size         print the allocated size of each file, in blocks\n\

```

```

"), stdout);
    fputs (_("\
-S          sort by file size, largest first\n\
"), stdout);
    fputs (_("\
--sort=WORD      sort by WORD instead of name: none (-U), size (-S)\\
,\n\
                           time (-t), version (-v), extension (-X), width\n\
\n\
"), stdout);
    fputs (_("\
--time=WORD      select which timestamp used to display or sort;\n\
access time (-u): atime, access, use;\n\
metadata change time (-c): ctime, status;\n\
modified time (default): mtime, modification;\n\
birth time: birth, creation;\n\
with -l, WORD determines which time to show;\n\
with --sort=time, sort by WORD (newest first)\n\
\n\
"), stdout);
    fputs (_("\
--time-style=TIME_STYLE\n\
                           time/date format with -l; see TIME_STYLE below\n\
"), stdout);
    fputs (_("\
-t          sort by time, newest first; see --time\n\
-T, --tabsize=COLS      assume tab stops at each COLS instead of 8\n\
"), stdout);
    fputs (_("\
-u          with -lt: sort by, and show, access time;\n\
               with -l: show access time and sort by name;\n\
               otherwise: sort by access time, newest first\n\
\n\
"), stdout);
    fputs (_("\
-U          do not sort; list entries in directory order\n\
"), stdout);
    fputs (_("\
-v          natural sort of (version) numbers within text\n\
"), stdout);
    fputs (_("\
-w, --width=COLS      set output width to COLS. 0 means no limit\n\
-x          list entries by lines instead of by columns\n\
-X          sort alphabetically by entry extension\n\
-Z, --context
    --zero      print any security context of each file\n\
               end each output line with NUL, not newline\n\
-1          list one file per line\n\
"), stdout);
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
    emit_size_note ();
    fputs (_("\
\n\
The TIME_STYLE argument can be full-iso, long-iso, iso, locale, or +FORMAT.\n\
FORMAT is interpreted like in date(1). If FORMAT is FORMAT1<newline>FORMAT2,\n\
then FORMAT1 applies to non-recent files and FORMAT2 to recent files.\n\
TIME_STYLE prefixed with 'posix-' takes effect only outside the POSIX locale.\n\
Also the TIME_STYLE environment variable sets the default style to use.\n\
"), stdout);
    fputs (_("\

```

```

\n\
The WHEN argument defaults to 'always' and can also be 'auto' or 'never'.\n\
"), stdout);
    fputs (_("\
\n\
Using color to distinguish file types is disabled both by default and\n\
with --color=never. With --color=auto, ls emits color codes only when\n\
standard output is connected to a terminal. The LS_COLORS environment\n\
variable can change the settings. Use the dircolors(1) command to set it.\n\
"), stdout);
    fputs (_("\
\n\
Exit status:\n\
0 if OK,\n\
1 if minor problems (e.g., cannot access subdirectory),\n\
2 if serious trouble (e.g., cannot access command-line argument).\n\
"), stdout);
    emit_ancillary_info (PROGRAM_NAME);
}
exit (status);
}
"""

,
"error_category" : "Platform-Specific Command Execution and Fork Handling",
"error" : "EDC5126I: Fork failed due to system limitations on z/OS, caused by using fork()\nand execvpe() without handling MVS-specific processes.",
"correct_code":
"""

/* 'dir', 'vdir' and 'ls' directory listing programs for GNU.
Copyright (C) 1985-2024 Free Software Foundation, Inc.
```

This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License  
along with this program. If not, see <<https://www.gnu.org/licenses/>>. \*/

/\* If ls\_mode is LS\_MULTI\_COL,  
the multi-column format is the default regardless  
of the type of output device.  
This is for the 'dir' program.

If ls\_mode is LS\_LONG\_FORMAT,  
the long format is the default regardless of the  
type of output device.  
This is for the 'vdir' program.

If ls\_mode is LS\_LS,  
the output format depends on whether the output  
device is a terminal.  
This is for the 'ls' program. \*/

/\* Written by Richard Stallman and David MacKenzie. \*/

```

/* Color support by Peter Anvin <Peter.Anvin@linux.org> and Dennis
Flaherty <dennif@denix.elk.miles.com> based on original patches by
Greg Lee <lee@uhunix.uhcc.hawaii.edu>. */

#include <config.h>
#include <ctype.h>
#include <sys/types.h>

#include <termios.h>
#if HAVE_STROPTS_H
# include <stropts.h>
#endif
#include <sys/ioctl.h>

#ifndef WINSIZE_IN_PTEM
# include <sys/stream.h>
# include <sys/ptem.h>
#endif

#include <stdio.h>
#include <setjmp.h>
#include <pwd.h>
#include <getopt.h>
#include <signal.h>
#include <selinux/selinux.h>
#include <uchar.h>

#if HAVE_LANGINFO_CODESET
# include <langinfo.h>
#endif

/* Use SA_NOCLDSTOP as a proxy for whether the sigaction machinery is
present. */
#ifndef SA_NOCLDSTOP
#define SA_NOCLDSTOP 0
#define sigprocmask(How, Set, Oset) /* empty */
#define sigset_t int
#if !HAVE_SIGINTERRUPT
#define siginterrupt(sig, flag) /* empty */
#endif
#endif

/* NonStop circa 2011 lacks both SA_RESTART and siginterrupt, so don't
restart syscalls after a signal handler fires. This may cause
colors to get messed up on the screen if 'ls' is interrupted, but
that's the best we can do on such a platform. */
#ifndef SA_RESTART
#define SA_RESTART 0
#endif

#include "system.h"
#include <fnmatch.h>

#include "acl.h"
#include "argmatch.h"
#include "assure.h"
#include "c-strcase.h"
#include "dev-ino.h"
#include "filenamecat.h"
#include "hard-locale.h"

```

```

#include "hash.h"
#include "human.h"
#include "filemode.h"
#include "filevercmp.h"
#include "idcache.h"
#include "ls.h"
#include "mbswidth.h"
#include "mpsort.h"
#include "obstack.h"
#include "quote.h"
#include "smack.h"
#include "stat-size.h"
#include "stat-time.h"
#include "strftime.h"
#include "xdecoint.h"
#include "xstrtol.h"
#include "xstrtol-error.h"
#include "areadlink.h"
#include "dircolors.h"
#include "xgethostname.h"
#include "c-ctype.h"
#include "canonicalize.h"
#include "statx.h"

#ifndef __MVS__
#define MVS_LS "/bin/ls"
#define MVS_LS_LEN (sizeof(MVS_LS)-1)
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

extern char** environ;
static bool
do_mvs_ls (int argc, char **argv)
{
    pid_t pid = fork();
    if (pid == 0) {
        argv[0] = MVS_LS;
        execvpe(MVS_LS, argv, environ);
    } else if (pid > 0) {
        int wstatus;
        pid_t waitchild = waitpid(pid, &wstatus, WUNTRACED | WCONTINUED);
        if (waitchild == -1) {
            perror("waitpid");
            return false;
        }
        if (WIFEXITED(wstatus)) {
            return WEXITSTATUS(wstatus) == 0 ? true : false;
        } else if (WIFSIGNALED(wstatus)) {
            fprintf(stderr, "killed by signal %d\n", WTERMSIG(wstatus));
        }
    }
}
#endif

/* Include <sys/capability.h> last to avoid a clash of <sys/types.h>
 * include guards with some premature versions of libcap.
 * For more details, see <https://bugzilla.redhat.com/483548>. */
#endif HAVE_CAP

```

```

#include <sys/capability.h>
#endif

#define PROGRAM_NAME (ls_mode == LS_LS ? "ls" \
: (ls_mode == LS_MULTI_COL \
? "dir" : "vdir"))

#define AUTHORS \
proper_name ("Richard M. Stallman"), \
proper_name ("David MacKenzie")

#define obstack_chunk_alloc malloc
#define obstack_chunk_free free

/* Unix-based readdir implementations have historically returned a dirent.d_ino
value that is sometimes not equal to the stat-obtained st_ino value for
that same entry. This error occurs for a readdir entry that refers
to a mount point. readdir's error is to return the inode number of
the underlying directory -- one that typically cannot be stat'ed, as
long as a file system is mounted on that directory. RELIABLE_D_INO
encapsulates whether we can use the more efficient approach of relying
on readdir-supplied d_ino values, or whether we must incur the cost of
calling stat or lstat to obtain each guaranteed-valid inode number. */

#ifndef REaddir_LIES_ABOUT_MOUNTPOINT_D_INO
#define REaddir_LIES_ABOUT_MOUNTPOINT_D_INO 1
#endif

#if REaddir_LIES_ABOUT_MOUNTPOINT_D_INO
#define RELIABLE_D_INO(dp) NOT_AN_INODE_NUMBER
#else
#define RELIABLE_D_INO(dp) D_INO (dp)
#endif

#if ! HAVE_STRUCT_STAT_ST_AUTHOR
#define st_author st_uid
#endif

enum filetype
{
    unknown,
    fifo,
    chardev,
    directory,
    blockdev,
    normal,
    symbolic_link,
    sock,
    whiteout,
    arg_directory
};

/* Display letters and indicators for each filetype.
Keep these in sync with enum filetype. */
static char const filetype_letter[] = "?pcdb-lswd";

/* Ensure that filetype and filetype_letter have the same
number of elements. */
static_assert (sizeof filetype_letter - 1 == arg_directory + 1);

```

```

#define FILETYPE_INDICATORS           \
{                                     \
    C_ORPHAN, C_FIFO, C_CHR, C_DIR, C_BLK, C_FILE, \
    C_LINK, C_SOCK, C_FILE, C_DIR               \
}

enum acl_type
{
    ACL_T_NONE,
    ACL_T_LSM_CONTEXT_ONLY,
    ACL_T_YES
};

struct fileinfo
{
    /* The file name. */
    char *name;

    /* For symbolic link, name of the file linked to, otherwise zero. */
    char *linkname;

    /* For terminal hyperlinks. */
    char *absolute_name;

    struct stat stat;

    enum filetype filetype;

    /* For symbolic link and long listing, st_mode of file linked to, otherwise
     * zero. */
    mode_t linkmode;

    /* security context. */
    char *scontext;

    bool stat_ok;

    /* For symbolic link and color printing, true if linked-to file
     * exists, otherwise false. */
    bool linkok;

    /* For long listings, true if the file has an access control list,
     * or a security context. */
    enum acl_type acl_type;

    /* For color listings, true if a regular file has capability info. */
    bool has_capability;

    /* Whether file name needs quoting. tri-state with -1 == unknown. */
    int quoted;

    /* Cached screen width (including quoting). */
    size_t width;
};

#define LEN_STR_PAIR(s) sizeof (s) - 1, s

/* Null is a valid character in a color indicator (think about Epson
printers, for example) so we have to use a length/buffer string
type. */

```

```

struct bin_str
{
    size_t len;           /* Number of bytes */
    char const *string;   /* Pointer to the same */
};

#ifndef HAVE_TCGETPGRP
#define tcgetpgrp(Fd) 0
#endif

static size_t quote_name (char const *name,
                         struct quoting_options const *options,
                         int needs_general_quoting,
                         const struct bin_str *color,
                         bool allow_pad, struct obstack *stack,
                         char const *absolute_name);
static size_t quote_name_buf (char **inbuf, size_t bufsize, char *name,
                             struct quoting_options const *options,
                             int needs_general_quoting, size_t *width,
                             bool *pad);
static int decode_switches (int argc, char **argv);
static bool file_ignored (char const *name);
static uintmax_t gobble_file (char const *name, enum filetype type,
                             ino_t inode, bool command_line_arg,
                             char const *dirname);
static const struct bin_str * get_color_indicator (const struct fileinfo *f,
                                                   bool symlink_target);
static bool print_color_indicator (const struct bin_str *ind);
static void put_indicator (const struct bin_str *ind);
static void add_ignore_pattern (char const *pattern);
static void attach (char *dest, char const *dirname, char const *name);
static void clear_files (void);
static void extract_dirs_from_files (char const *dirname,
                                    bool command_line_arg);
static void get_link_name (char const *filename, struct fileinfo *f,
                           bool command_line_arg);
static void indent (size_t from, size_t to);
static size_t calculate_columns (bool by_columns);
static void print_current_files (void);
static void print_dir (char const *name, char const *realname,
                      bool command_line_arg);
static size_t print_file_name_and_frills (const struct fileinfo *f,
                                           size_t start_col);
static void print_horizontal (void);
static int format_user_width (uid_t u);
static int format_group_width (gid_t g);
static void print_long_format (const struct fileinfo *f);
static void print_many_per_line (void);
static size_t print_name_with_quoting (const struct fileinfo *f,
                                       bool symlink_target,
                                       struct obstack *stack,
                                       size_t start_col);
static void prep_non_filename_text (void);
static bool print_type_indicator (bool stat_ok, mode_t mode,
                                 enum filetype type);
static void print_with_separator (char sep);
static void queue_directory (char const *name, char const *realname,
                            bool command_line_arg);
static void sort_files (void);

```

```

static void parse_ls_color (void);

static int getenv_quoting_style (void);

static size_t quote_name_width (char const *name,
                               struct quoting_options const *options,
                               int needs_general_quoting);

/* Initial size of hash table.
Most hierarchies are likely to be shallower than this. */
enum { INITIAL_TABLE_SIZE = 30 };

/* The set of 'active' directories, from the current command-line argument
to the level in the hierarchy at which files are being listed.
A directory is represented by its device and inode numbers (struct dev_ino).
A directory is added to this set when ls begins listing it or its
entries, and it is removed from the set just after ls has finished
processing it. This set is used solely to detect loops, e.g., with
mkdir loop; cd loop; ln -s ..//loop sub; ls -RL */
static Hash_table *active_dir_set;

#define LOOP_DETECT (!!active_dir_set)

/* The table of files in the current directory:

'cwd_file' points to a vector of 'struct fileinfo', one per file.
'cwd_n_alloc' is the number of elements space has been allocated for.
'cwd_n_used' is the number actually in use. */

/* Address of block containing the files that are described. */
static struct fileinfo * cwd_file;

/* Length of block that 'cwd_file' points to, measured in files. */
static size_t cwd_n_alloc;

/* Index of first unused slot in 'cwd_file'. */
static size_t cwd_n_used;

/* Whether files need padding due to quoting. */
static bool cwd_some_quoted;

/* Whether quoting style _may_ add outer quotes,
and whether aligning those is useful. */
static bool align_variable_outer_quotes;

/* Vector of pointers to files, in proper sorted order, and the number
of entries allocated for it. */
static void **sorted_file;
static size_t sorted_file_alloc;

/* When true, in a color listing, color each symlink name according to the
type of file it points to. Otherwise, color them according to the 'In'
directive in LS_COLORS. Dangling (orphan) symlinks are treated specially,
regardless. This is set when 'In=target' appears in LS_COLORS. */

static bool color_symlink_as_referent;

static char const *hostname;

/* Mode of appropriate file for coloring. */

```

```

static mode_t
file_or_link_mode (struct fileinfo const *file)
{
    return (color_symlink_as_referent && file->linkok
            ? file->linkmode : file->stat.st_mode);
}

/* Record of one pending directory waiting to be listed. */

struct pending
{
    char *name;
    /* If the directory is actually the file pointed to by a symbolic link we
     were told to list, 'realname' will contain the name of the symbolic
     link, otherwise zero. */
    char *realname;
    bool command_line_arg;
    struct pending *next;
};

static struct pending *pending_dirs;

/* Current time in seconds and nanoseconds since 1970, updated as
needed when deciding whether a file is recent. */

static struct timespec current_time;

static bool print_scontext;
static char UNKNOWN_SECURITY_CONTEXT[] = "?";

/* Whether any of the files has an ACL. This affects the width of the
mode column. */

static bool any_has_acl;

/* The number of columns to use for columns containing inode numbers,
block sizes, link counts, owners, groups, authors, major device
numbers, minor device numbers, and file sizes, respectively. */

static int inode_number_width;
static int block_size_width;
static int nlink_width;
static int scontext_width;
static int owner_width;
static int group_width;
static int author_width;
static int major_device_number_width;
static int minor_device_number_width;
static int file_size_width;

/* Option flags */

/* long_format for lots of info, one per line.
one_per_line for just names, one per line.
many_per_line for just names, many per line, sorted vertically.
horizontal for just names, many per line, sorted horizontally.
with_commas for just names, many per line, separated by commas.

-I (and other options that imply -l), -1, -C, -x and -m control

```

```

this parameter. */

enum format
{
    long_format,           /* -l and other options that imply -l */
    one_per_line,          /* -1 */
    many_per_line,         /* -C */
    horizontal,            /* -x */
    with_commas,           /* -m */
};

static enum format format;

/* 'full-iso' uses full ISO-style dates and times. 'long-iso' uses longer
ISO-style timestamps, though shorter than 'full-iso'. 'iso' uses shorter
ISO-style timestamps. 'locale' uses locale-dependent timestamps. */
enum time_style
{
    full_iso_time_style, /* --time-style=full-iso */
    long_iso_time_style, /* --time-style=long-iso */
    iso_time_style,      /* --time-style=iso */
    locale_time_style,   /* --time-style=locale */
};

static char const *const time_style_args[] =
{
    "full-iso", "long-iso", "iso", "locale", nullptr
};
static enum time_style const time_style_types[] =
{
    full_iso_time_style, long_iso_time_style, iso_time_style,
    locale_time_style
};
ARGMATCH_VERIFY (time_style_args, time_style_types);

/* Type of time to print or sort by. Controlled by -c and -u.
The values of each item of this enum are important since they are
used as indices in the sort functions array (see sort_files()). */

enum time_type
{
    time_mtime = 0,        /* default */
    time_ctime,             /* -c */
    time_atime,             /* -u */
    time_btime,             /* birth time */
    time_numtypes           /* the number of elements of this enum */
};

static enum time_type time_type;

/* The file characteristic to sort by. Controlled by -t, -S, -U, -X, -v.
The values of each item of this enum are important since they are
used as indices in the sort functions array (see sort_files()). */

enum sort_type
{
    sort_name = 0,           /* default */
    sort_extension,          /* -X */
    sort_width,              /* -U */
    sort_size,               /* -S */
};

```

```

sort_version,           /* -v */
sort_time,              /* -t; must be second to last */
sort_none,               /* -U; must be last */
sort_numtypes           /* the number of elements of this enum */
};

static enum sort_type sort_type;

/* Direction of sort.
false means highest first if numeric,
lowest first if alphabetic;
these are the defaults.
true means the opposite order in each case. -r */

static bool sort_reverse;

/* True means to display owner information. -g turns this off. */

static bool print_owner = true;

/* True means to display author information. */

static bool print_author;

/* True means to display group information. -G and -o turn this off. */

static bool print_group = true;

/* True means print the user and group id's as numbers rather
than as names. -n */

static bool numeric_ids;

/* True means mention the size in blocks of each file. -s */

static bool print_block_size;

/* Human-readable options for output, when printing block counts. */
static int human_output_opts;

/* The units to use when printing block counts. */
static uintmax_t output_block_size;

/* Likewise, but for file sizes. */
static int file_human_output_opts;
static uintmax_t file_output_block_size = 1;

/* Follow the output with a special string. Using this format,
Emacs' dired mode starts up twice as fast, and can handle all
strange characters in file names. */
static bool dired;

/* 'none' means don't mention the type of files.
'slash' means mention directories only, with a '/'.
'file_type' means mention file types.
'classify' means mention file types and mark executables.

Controlled by -F, -p, and --indicator-style. */

enum indicator_style

```

```

{
    none = 0, /* --indicator-style=none (default) */
    slash,    /* -p, --indicator-style=slash */
    file_type, /* --indicator-style=file-type */
    classify   /* -F, --indicator-style=classify */
};

static enum indicator_style indicator_style;

/* Names of indicator styles. */
static char const *const indicator_style_args[] =
{
    "none", "slash", "file-type", "classify", nullptr
};
static enum indicator_style const indicator_style_types[] =
{
    none, slash, file_type, classify
};
ARGMATCH_VERIFY(indicator_style_args, indicator_style_types);

/* True means use colors to mark types. Also define the different
colors as well as the stuff for the LS_COLORS environment variable.
The LS_COLORS variable is now in a termcap-like format. */

static bool print_with_color;

static bool print_hyperlink;

/* Whether we used any colors in the output so far. If so, we will
need to restore the default color later. If not, we will need to
call prep_non_filename_text before using color for the first time. */

static bool used_color = false;

enum when_type
{
    when_never,           /* 0: default or --color=never */
    when_always,          /* 1: --color=always */
    when_if_tty           /* 2: --color=tty */
};

enum Dereference_symlink
{
    DEREF_UNDEFINED = 0,      /* default */
    DEREF_NEVER,
    DEREF_COMMAND_LINE_ARGUMENTS, /* -H */
    DEREF_COMMAND_LINE_SYMLINK_TO_DIR, /* the default, in certain cases */
    DEREF_ALWAYS           /* -L */
};

enum indicator_no
{
    C_LEFT, C_RIGHT, C_END, C_RESET, C_NORM, C_FILE, C_DIR, C_LINK,
    C_FIFO, C_SOCK,
    C_BLK, C_CHR, C_MISSING, C_ORPHAN, C_EXEC, C_DOOR, C_SETUID, C_SETGID,
    C_STICKY, C_OTHER_WRITABLE, C_STICKY_OTHER_WRITABLE, C_CAP,
    C_MULTIHARDLINK,
    C_CLR_TO_EOL
};

```

```

static char const *const indicator_name[]=
{
    "lc", "rc", "ec", "rs", "no", "fi", "di", "ln", "pi", "so",
    "bd", "cd", "mi", "or", "ex", "do", "su", "sg", "st",
    "ow", "tw", "ca", "mh", "cl", nullptr
};

struct color_ext_type
{
    struct bin_str ext;          /* The extension we're looking for */
    struct bin_str seq;          /* The sequence to output when we do */
    bool exact_match;            /* Whether to compare case insensitively */
    struct color_ext_type *next; /* Next in list */
};

static struct bin_str color_indicator[] =
{
    { LEN_STR_PAIR ("\033[") },           /* lc: Left of color sequence */
    { LEN_STR_PAIR ("m") },                /* rc: Right of color sequence */
    { 0, nullptr },                      /* ec: End color (replaces lc+rs+rc) */
    { LEN_STR_PAIR ("0") },                /* rs: Reset to ordinary colors */
    { 0, nullptr },                      /* no: Normal */
    { 0, nullptr },                      /* fi: File: default */
    { LEN_STR_PAIR ("01;34") },           /* di: Directory: bright blue */
    { LEN_STR_PAIR ("01;36") },           /* ln: Symlink: bright cyan */
    { LEN_STR_PAIR ("33") },              /* pi: Pipe: yellow/brown */
    { LEN_STR_PAIR ("01;35") },           /* so: Socket: bright magenta */
    { LEN_STR_PAIR ("01;33") },           /* bd: Block device: bright yellow */
    { LEN_STR_PAIR ("01;33") },           /* cd: Char device: bright yellow */
    { 0, nullptr },                      /* mi: Missing file: undefined */
    { 0, nullptr },                      /* or: Orphaned symlink: undefined */
    { LEN_STR_PAIR ("01;32") },           /* ex: Executable: bright green */
    { LEN_STR_PAIR ("01;35") },           /* do: Door: bright magenta */
    { LEN_STR_PAIR ("37;41") },           /* su: setuid: white on red */
    { LEN_STR_PAIR ("30;43") },           /* sg: setgid: black on yellow */
    { LEN_STR_PAIR ("37;44") },           /* st: sticky: black on blue */
    { LEN_STR_PAIR ("34;42") },           /* ow: other-writable: blue on green */
    { LEN_STR_PAIR ("30;42") },           /* tw: ow w/ sticky: black on green */
    { 0, nullptr },                      /* ca: disabled by default */
    { 0, nullptr },                      /* mh: disabled by default */
    { LEN_STR_PAIR ("\033[K") },          /* cl: clear to end of line */
};

/* A list mapping file extensions to corresponding display sequence. */
static struct color_ext_type *color_ext_list = nullptr;

/* Buffer for color sequences */
static char *color_buf;

/* True means to check for orphaned symbolic link, for displaying
colors, or to group symlink to directories with other dirs. */

static bool check_symlink_mode;

/* True means mention the inode number of each file. -i */

static bool print_inode;

/* What to do with symbolic links. Affected by -d, -F, -H, -l (and
other options that imply -l), and -L. */

```

```

static enum Dereference_symlink dereference;

/* True means when a directory is found, display info on its
contents. -R */

static bool recursive;

/* True means when an argument is a directory name, display info
on it itself. -d */

static bool immediate_dirs;

/* True means that directories are grouped before files. */

static bool directories_first;

/* Which files to ignore. */

static enum
{
    /* Ignore files whose names start with '.', and files specified by
     --hide and --ignore. */
    IGNORE_DEFAULT = 0,
    /* Ignore '..', and files specified by --ignore. */
    IGNORE_DOT_AND_DOTDOT,
    /* Ignore only files specified by --ignore. */
    IGNORE_MINIMAL
} ignore_mode;

/* A linked list of shell-style globbing patterns. If a non-argument
file name matches any of these patterns, it is ignored.
Controlled by -I. Multiple -I options accumulate.
The -B option adds '*~' and '.*~' to this list. */

struct ignore_pattern
{
    char const *pattern;
    struct ignore_pattern *next;
};

static struct ignore_pattern *ignore_patterns;

/* Similar to IGNORE_PATTERNS, except that -a or -A causes this
variable itself to be ignored. */
static struct ignore_pattern *hide_patterns;

/* True means output nongraphic chars in file names as '?'.
(-q, --hide-control-chars)
qmark_funny_chars and the quoting style (-Q, --quoting-style=WORD) are
independent. The algorithm is: first, obey the quoting style to get a
string representing the file name; then, if qmark_funny_chars is set,
replace all nonprintable chars in that string with '?'. It's necessary
to replace nonprintable chars even in quoted strings, because we don't
want to mess up the terminal if control chars get sent to it, and some
quoting methods pass through control chars as-is. */
static bool qmark_funny_chars;

```

```

/* Quoting options for file and dir name output. */

static struct quoting_options *filename_quoting_options;
static struct quoting_options *dirname_quoting_options;

/* The number of chars per hardware tab stop. Setting this to zero
inhibits the use of TAB characters for separating columns. -T */
static size_t tabsize;

/* True means print each directory name before listing it. */

static bool print_dir_name;

/* The line length to use for breaking lines in many-per-line format.
Can be set with -w. If zero, there is no limit. */

static size_t line_length;

/* The local time zone rules, as per the TZ environment variable. */

static timezone_t localtz;

/* If true, the file listing format requires that stat be called on
each file. */

static bool format_needs_stat;

/* Similar to 'format_needs_stat', but set if only the file type is
needed. */

static bool format_needs_type;

/* An arbitrary limit on the number of bytes in a printed timestamp.
This is set to a relatively small value to avoid the need to worry
about denial-of-service attacks on servers that run "ls" on behalf
of remote clients. 1000 bytes should be enough for any practical
timestamp format. */

enum { TIME_STAMP_LEN_MAXIMUM = MAX (1000, INT_STRLEN_BOUND (time_t)) };

/* strftime formats for non-recent and recent files, respectively, in
-l output. */

static char const *long_time_format[2] =
{
    /* strftime format for non-recent files (older than 6 months), in
    -l output. This should contain the year, month and day (at
    least), in an order that is understood by people in your
    locale's territory. Please try to keep the number of used
    screen columns small, because many people work in windows with
    only 80 columns. But make this as wide as the other string
    below, for recent files. */
    /* TRANSLATORS: ls output needs to be aligned for ease of reading,
    so be wary of using variable width fields from the locale.
    Note %b is handled specially by ls and aligned correctly.
    Note also that specifying a width as in %5b is erroneous as strftime
    will count bytes rather than characters in multibyte locales. */
    N_(""%b %e %Y"),
    /* strftime format for recent files (younger than 6 months), in -l
    output. This should contain the month, day and time (at

```

```

least), in an order that is understood by people in your
locale's territory. Please try to keep the number of used
screen columns small, because many people work in windows with
only 80 columns. But make this as wide as the other string
above, for non-recent files. */
/* TRANSLATORS: ls output needs to be aligned for ease of reading,
so be wary of using variable width fields from the locale.
Note %b is handled specially by ls and aligned correctly.
Note also that specifying a width as in %5b is erroneous as strftime
will count bytes rather than characters in multibyte locales. */
N_(""%b %e %H:%M")
};

/* The set of signals that are caught. */
static sigset_t caught_signals;

/* If nonzero, the value of the pending fatal signal. */
static sig_atomic_t volatile interrupt_signal;

/* A count of the number of pending stop signals that have been received. */
static sig_atomic_t volatile stop_signal_count;

/* Desired exit status. */
static int exit_status;

/* Exit statuses. */
enum
{
    /* "ls" had a minor problem. E.g., while processing a directory,
    ls obtained the name of an entry via readdir, yet was later
    unable to stat that name. This happens when listing a directory
    in which entries are actively being removed or renamed. */
    LS_MINOR_PROBLEM = 1,
    /* "ls" had more serious trouble (e.g., memory exhausted, invalid
    option or failure to stat a command line argument. */
    LS_FAILURE = 2
};

/* For long options that have no equivalent short option, use a
non-character as a pseudo short option, starting with CHAR_MAX + 1. */
enum
{
    AUTHOR_OPTION = CHAR_MAX + 1,
    BLOCK_SIZE_OPTION,
    COLOR_OPTION,
    DEREference_COMMAND_LINE_SYMLINK_TO_DIR_OPTION,
    FILE_TYPE_INDICATOR_OPTION,
    FORMAT_OPTION,
    FULL_TIME_OPTION,
    GROUP_DIRECTORIES_FIRST_OPTION,
    HIDE_OPTION,
    HYPERLINK_OPTION,
    INDICATOR_STYLE_OPTION,
    QUOTING_STYLE_OPTION,
    SHOW_CONTROL_CHARS_OPTION,

```

```

SI_OPTION,
SORT_OPTION,
TIME_OPTION,
TIME_STYLE_OPTION,
ZERO_OPTION,
};

static struct option const long_options[] =
{
{"all", no_argument, nullptr, 'a'},
 {"escape", no_argument, nullptr, 'b'},
 {"directory", no_argument, nullptr, 'd'},
 {"dired", no_argument, nullptr, 'D'},
 {"full-time", no_argument, nullptr, FULL_TIME_OPTION},
 {"group-directories-first", no_argument, nullptr,
 GROUP_DIRECTORIES_FIRST_OPTION},
 {"human-readable", no_argument, nullptr, 'h'},
 {"inode", no_argument, nullptr, 'i'},
 {"kibibytes", no_argument, nullptr, 'k'},
 {"numeric-uid-gid", no_argument, nullptr, 'n'},
 {"no-group", no_argument, nullptr, 'G'},
 {"hide-control-chars", no_argument, nullptr, 'q'},
 {"reverse", no_argument, nullptr, 'r'},
 {"size", no_argument, nullptr, 's'},
 {"width", required_argument, nullptr, 'w'},
 {"almost-all", no_argument, nullptr, 'A'},
 {"ignore-backups", no_argument, nullptr, 'B'},
 {"classify", optional_argument, nullptr, 'F'},
 {"file-type", no_argument, nullptr, FILE_TYPE_INDICATOR_OPTION},
 {"si", no_argument, nullptr, SI_OPTION},
 {"dereference-command-line", no_argument, nullptr, 'H'},
 {"dereference-command-line-symlink-to-dir", no_argument, nullptr,
 DEREference_COMMAND_LINE_SYMLINK_TO_DIR_OPTION},
 {"hide", required_argument, nullptr, HIDE_OPTION},
 {"ignore", required_argument, nullptr, 'I'},
 {"indicator-style", required_argument, nullptr, INDICATOR_STYLE_OPTION},
 {"dereference", no_argument, nullptr, 'L'},
 {"literal", no_argument, nullptr, 'N'},
 {"quote-name", no_argument, nullptr, 'Q'},
 {"quoting-style", required_argument, nullptr, QUOTING_STYLE_OPTION},
 {"recursive", no_argument, nullptr, 'R'},
 {"format", required_argument, nullptr, FORMAT_OPTION},
 {"show-control-chars", no_argument, nullptr, SHOW_CONTROL_CHARS_OPTION},
 {"sort", required_argument, nullptr, SORT_OPTION},
 {"tabsize", required_argument, nullptr, 'T'},
 {"time", required_argument, nullptr, TIME_OPTION},
 {"time-style", required_argument, nullptr, TIME_STYLE_OPTION},
 {"zero", no_argument, nullptr, ZERO_OPTION},
 {"color", optional_argument, nullptr, COLOR_OPTION},
 {"hyperlink", optional_argument, nullptr, HYPERLINK_OPTION},
 {"block-size", required_argument, nullptr, BLOCK_SIZE_OPTION},
 {"context", no_argument, 0, 'Z'},
 {"author", no_argument, nullptr, AUTHOR_OPTION},
 {GETOPT_HELP_OPTION_DECL},
 {GETOPT_VERSION_OPTION_DECL},
 {nullptr, 0, nullptr, 0}
};

static char const *const format_args[] =
{

```

```

"verbose", "long", "commas", "horizontal", "across",
"vertical", "single-column", nullptr
};
static enum format const format_types[] =
{
long_format, long_format, with_commas, horizontal, horizontal,
many_per_line, one_per_line
};
ARGMATCH_VERIFY (format_args, format_types);

static char const *const sort_args[] =
{
"none", "time", "size", "extension", "version", "width", nullptr
};
static enum sort_type const sort_types[] =
{
sort_none, sort_time, sort_size, sort_extension, sort_version, sort_width
};
ARGMATCH_VERIFY (sort_args, sort_types);

static char const *const time_args[] =
{
"atime", "access", "use",
"ctime", "status",
"mtime", "modification",
"birth", "creation",
nullptr
};
static enum time_type const time_types[] =
{
time_atime, time_atime, time_atime,
time_ctime, time_ctime,
time_mtime, time_mtime,
time_btime, time_btime,
};
ARGMATCH_VERIFY (time_args, time_types);

static char const *const when_args[] =
{
/* force and none are for compatibility with another color-ls version */
"always", "yes", "force",
"never", "no", "none",
"auto", "tty", "if-tty", nullptr
};
static enum when_type const when_types[] =
{
when_always, when_always, when_always,
when_never, when_never, when_never,
when_if_tty, when_if_tty, when_if_tty
};
ARGMATCH_VERIFY (when_args, when_types);

/* Information about filling a column. */
struct column_info
{
bool valid_len;
size_t line_len;
size_t *col_arr;
};

```

```

/* Array with information about column fullness. */
static struct column_info *column_info;

/* Maximum number of columns ever possible for this display. */
static size_t max_idx;

/* The minimum width of a column is 3: 1 character for the name and 2
for the separating white space. */
enum { MIN_COLUMN_WIDTH = 3 };

/* This zero-based index is for the --dired option. It is incremented
for each byte of output generated by this program so that the beginning
and ending indices (in that output) of every file name can be recorded
and later output themselves. */
static off_t dired_pos;

static void
dired_outbyte (char c)
{
dired_pos++;
putchar (c);
}

/* Output the buffer S, of length S_LEN, and increment DIRED_POS by S_LEN. */
static void
dired_outbuf (char const *s, size_t s_len)
{
dired_pos += s_len;
fwrite (s, sizeof *s, s_len, stdout);
}

/* Output the string S, and increment DIRED_POS by its length. */
static void
dired_outstring (char const *s)
{
dired_outbuf (s, strlen (s));
}

static void
dired_indent (void)
{
if (dired)
  dired_outstring (" ");
}

/* With --dired, store pairs of beginning and ending indices of file names. */
static struct obstack dired_obstack;

/* With --dired, store pairs of beginning and ending indices of any
directory names that appear as headers (just before 'total' line)
for lists of directory entries. Such directory names are seen when
listing hierarchies using -R and when a directory is listed with at
least one other command line argument. */
static struct obstack subdired_obstack;

/* Save the current index on the specified obstack, OBS. */
static void
push_current_dired_pos (struct obstack *obs)
{

```

```

if (dired)
    obstack_grow (obs, &dired_pos, sizeof dired_pos);
}

/* With -R, this stack is used to help detect directory cycles.
The device/inode pairs on this stack mirror the pairs in the
active_dir_set hash table. */
static struct obstack dev_ino_obstack;

/* Push a pair onto the device/inode stack. */
static void
dev_ino_push (dev_t dev, ino_t ino)
{
void *vdi;
struct dev_ino *di;
int dev_ino_size = sizeof *di;
obstack_blank (&dev_ino_obstack, dev_ino_size);
vdi = obstack_next_free (&dev_ino_obstack);
di = vdi;
di--;
di->st_dev = dev;
di->st_ino = ino;
}

/* Pop a dev/ino struct off the global dev_ino_obstack
and return that struct. */
static struct dev_ino
dev_ino_pop (void)
{
void *vdi;
struct dev_ino *di;
int dev_ino_size = sizeof *di;
affirm (dev_ino_size <= obstack_object_size (&dev_ino_obstack));
obstack_blank_fast (&dev_ino_obstack, -dev_ino_size);
vdi = obstack_next_free (&dev_ino_obstack);
di = vdi;
return *di;
}

static void
assert_matching_dev_ino (char const *name, struct dev_ino di)
{
MAYBE_UNUSED struct stat sb;
assure (0 <= stat (name, &sb));
assure (sb.st_dev == di.st_dev);
assure (sb.st_ino == di.st_ino);
}

static char eolbyte = '\n';

/* Write to standard output PREFIX, followed by the quoting style and
a space-separated list of the integers stored in OS all on one line. */

static void
dired_dump_obstack (char const *prefix, struct obstack *os)
{
size_t n_pos;

n_pos = obstack_object_size (os) / sizeof (dired_pos);
if (n_pos > 0)

```

```

{
off_t *pos = obstack_finish (os);
fputs (prefix, stdout);
for (size_t i = 0; i < n_pos; i++)
{
    intmax_t p = pos[i];
    printf (" %jd", p);
}
putchar ('\n');
}

/* Return the platform birthtime member of the stat structure,
or fallback to the mtime member, which we have populated
from the statx structure or reset to an invalid timestamp
where birth time is not supported. */
static struct timespec
get_stat_btime (struct stat const *st)
{
struct timespec btimespec;

#if HAVE_STATX && defined STATX_INO
btimespec = get_stat_mtime (st);
#else
btimespec = get_stat_birthtime (st);
#endif

return btimespec;
}

#if HAVE_STATX && defined STATX_INO
ATTRIBUTE_PURE
static unsigned int
time_type_to_statx (void)
{
switch (time_type)
{
    case time_ctime:
        return STATX_CTIME;
    case time_mtime:
        return STATX_MTIME;
    case time_atime:
        return STATX_ATIME;
    case time_btime:
        return STATX_BTIME;
    default:
        unreachable ();
}
return 0;
}

ATTRIBUTE_PURE
static unsigned int
calc_req_mask (void)
{
unsigned int mask = STATX_MODE;

if (print_inode)
    mask |= STATX_INO;

```

```

if (print_block_size)
    mask |= STATX_BLOCKS;

if (format == long_format) {
    mask |= STATX_NLINK | STATX_SIZE | time_type_to_statx ();
    if (print_owner || print_author)
        mask |= STATX_UID;
    if (print_group)
        mask |= STATX_GID;
}

switch (sort_type)
{
case sort_none:
case sort_name:
case sort_version:
case sort_extension:
case sort_width:
break;
case sort_time:
mask |= time_type_to_statx ();
break;
case sort_size:
mask |= STATX_SIZE;
break;
default:
unreachable ();
}

return mask;
}

static int
do_statx (int fd, char const *name, struct stat *st, int flags,
          unsigned int mask)
{
struct statx stx;
bool want_btime = mask & STATX_BTIME;
int ret = statx (fd, name, flags | AT_NO_AUTOMOUNT, mask, &stx);
if (ret >= 0)
{
    statx_to_stat (&stx, st);
    /* Since we only need one timestamp type,
       store birth time in st_mtim. */
    if (want_btime)
    {
        if (stx.stx_mask & STATX_BTIME)
            st->st_mtim = statx_timestamp_to_timespec (stx.stx_btime);
        else
            st->st_mtim.tv_sec = st->st_mtim.tv_nsec = -1;
    }
}
return ret;
}

static int
do_stat (char const *name, struct stat *st)
{
return do_statx (AT_FDCWD, name, st, 0, calc_req_mask ());
}

```

```

}

static int
do_lstat (char const *name, struct stat *st)
{
return do_statx (AT_FDCWD, name, st, AT_SYMLINK_NOFOLLOW, calc_req_mask ());
}

static int
stat_for_mode (char const *name, struct stat *st)
{
return do_statx (AT_FDCWD, name, st, 0, STATX_MODE);
}

/* dev+ino should be static, so no need to sync with backing store */
static int
stat_for_ino (char const *name, struct stat *st)
{
return do_statx (AT_FDCWD, name, st, 0, STATX_INO);
}

static int
fstat_for_ino (int fd, struct stat *st)
{
return do_statx (fd, "", st, AT_EMPTY_PATH, STATX_INO);
}
#else
static int
do_stat (char const *name, struct stat *st)
{
return stat (name, st);
}

static int
do_lstat (char const *name, struct stat *st)
{
return lstat (name, st);
}

static int
stat_for_mode (char const *name, struct stat *st)
{
return stat (name, st);
}

static int
stat_for_ino (char const *name, struct stat *st)
{
return stat (name, st);
}

static int
fstat_for_ino (int fd, struct stat *st)
{
return fstat (fd, st);
}
#endif

/* Return the address of the first plain %b spec in FMT, or nullptr if
there is no such spec. %5b etc. do not match, so that user
```

```

widths/flags are honored. */

ATTRIBUTE_PURE
static char const *
first_percent_b (char const *fmt)
{
for (; *fmt; fmt++)
    if (fmt[0] == '%')
        switch (fmt[1])
        {
        case 'b': return fmt;
        case '%': fmt++; break;
        }
return nullptr;
}

static char RFC3986[256];
static void
file_escape_init (void)
{
for (int i = 0; i < 256; i++)
    RFC3986[i] |= c_isalnum (i) || i == '~' || i == '-' || i == '.' || i == '_';
}

enum { MBSWIDTH_FLAGS = MBSW_REJECT_INVALID |
MBSW_REJECT_UNPRINTABLE };

/* Read the abbreviated month names from the locale, to align them
and to determine the max width of the field and to truncate names
greater than our max allowed.
Note even though this handles multibyte locales correctly
it's not restricted to them as single byte locales can have
variable width abbreviated months and also precomputing/caching
the names was seen to increase the performance of ls significantly. */

/* abformat[RECENT][MON] is the format to use for timestamps with
recentness RECENT and month MON. */
enum { ABFORMAT_SIZE = 128 };
static char abformat[2][12][ABFORMAT_SIZE];
/* True if precomputed formats should be used. This can be false if
nl_langinfo fails, if a format or month abbreviation is unusually
long, or if a month abbreviation contains '%'. */
static bool use_abformat;

/* Store into ABMON the abbreviated month names, suitably aligned.
Return true if successful. */

static bool
abmon_init (char abmon[12][ABFORMAT_SIZE])
{
#ifndef HAVE_NL_LANGINFO
return false;
#else
int max_mon_width = 0;
int mon_width[12];
int mon_len[12];

for (int i = 0; i < 12; i++)
{
    char const *abbr = nl_langinfo (ABMON_1 + i);
    if (abbr[0] != '%')
        mon_width[i] = strlen (abbr);
    else
        mon_width[i] = 1;
    if (max_mon_width < mon_width[i])
        max_mon_width = mon_width[i];
}
for (int i = 0; i < 12; i++)
    abmon[i] = malloc (max_mon_width + 1);
    if (!abmon[i])
        return false;
    abmon[i][max_mon_width] = '\0';
}

```

```

mon_len[i] = strnlen (abbr, ABFORMAT_SIZE);
if (mon_len[i] == ABFORMAT_SIZE)
    return false;
if (strchr (abbr, '%'))
    return false;
mon_width[i] = mbswidth (strcpy (abmon[i], abbr), MBSWIDTH_FLAGS);
if (mon_width[i] < 0)
    return false;
max_mon_width = MAX (max_mon_width, mon_width[i]);
}

for (int i = 0; i < 12; i++)
{
    int fill = max_mon_width - mon_width[i];
    if (ABFORMAT_SIZE - mon_len[i] <= fill)
        return false;
    bool align_left = !isdigit (to_uchar (abmon[i][0]));
    int fill_offset;
    if (align_left)
        fill_offset = mon_len[i];
    else
    {
        memmove (abmon[i] + fill, abmon[i], mon_len[i]);
        fill_offset = 0;
    }
    memset (abmon[i] + fill_offset, ' ', fill);
    abmon[i][mon_len[i] + fill] = '\0';
}

return true;
#endif
}

/* Initialize ABFORMAT and USE_ABFORMAT. */

static void
abformat_init (void)
{
    char const *pb[2];
    for (int recent = 0; recent < 2; recent++)
        pb[recent] = first_percent_b (long_time_format[recent]);
    if (! (pb[0] || pb[1]))
        return;

    char abmon[12][ABFORMAT_SIZE];
    if (! abmon_init (abmon))
        return;

    for (int recent = 0; recent < 2; recent++)
    {
        char const *fmt = long_time_format[recent];
        for (int i = 0; i < 12; i++)
        {
            char *nfmt = abformat[recent][i];
            int nbytes;

            if (! pb[recent])
                nbytes = snprintf (nfmt, ABFORMAT_SIZE, "%s", fmt);
            else
            {

```

```

        if (! (pb[recent] - fmt <= MIN (ABFORMAT_SIZE, INT_MAX)))
            return;
        int prefix_len = pb[recent] - fmt;
        nbytes = snprintf (nfmt, ABFORMAT_SIZE, "%.*s%s%s",
                           prefix_len, fmt, abmon[i], pb[recent] + 2);
    }

    if (! (0 <= nbytes && nbytes < ABFORMAT_SIZE))
        return;
}

use_abformat = true;
}

static size_t
dev_ino_hash (void const *x, size_t table_size)
{
    struct dev_ino const *p = x;
    return (uintmax_t) p->st_ino % table_size;
}

static bool
dev_ino_compare (void const *x, void const *y)
{
    struct dev_ino const *a = x;
    struct dev_ino const *b = y;
    return PSAME_INODE (a, b);
}

static void
dev_ino_free (void *x)
{
    free (x);
}

/* Add the device/inode pair (P->st_dev/P->st_ino) to the set of
active directories. Return true if there is already a matching
entry in the table. */

static bool
visit_dir (dev_t dev, ino_t ino)
{
    struct dev_ino *ent;
    struct dev_ino *ent_from_table;
    bool found_match;

    ent = xmalloc (sizeof *ent);
    ent->st_ino = ino;
    ent->st_dev = dev;

    /* Attempt to insert this entry into the table. */
    ent_from_table = hash_insert (active_dir_set, ent);

    if (ent_from_table == nullptr)
    {
        /* Insertion failed due to lack of memory. */
        xalloc_die ();
    }
}

```

```

found_match = (ent_from_table != ent);

if (found_match)
{
    /* ent was not inserted, so free it. */
    free (ent);
}

return found_match;
}

static void
free_pending_ent (struct pending *p)
{
free (p->name);
free (p->realname);
free (p);
}

static bool
is_colored (enum indicator_no type)
{
size_t len = color_indicator[type].len;
char const *s = color_indicator[type].string;
return ! (len == 0)
    || (len == 1 && STRNCMP_LIT (s, "0") == 0)
    || (len == 2 && STRNCMP_LIT (s, "00") == 0));
}

static void
restore_default_color (void)
{
put_indicator (&color_indicator[C_LEFT]);
put_indicator (&color_indicator[C_RIGHT]);
}

static void
set_normal_color (void)
{
if (print_with_color && is_colored (C_NORM))
{
    put_indicator (&color_indicator[C_LEFT]);
    put_indicator (&color_indicator[C_NORM]);
    put_indicator (&color_indicator[C_RIGHT]);
}
}

/* An ordinary signal was received; arrange for the program to exit. */

static void
sighandler (int sig)
{
if (! SA_NOCLDSTOP)
    signal (sig, SIG_IGN);
if (! interrupt_signal)
    interrupt_signal = sig;
}

/* A SIGHSTP was received; arrange for the program to suspend itself. */

```

```

static void
stophandler (int sig)
{
if (! SA_NOCLDSTOP)
    signal (sig, stophandler);
if (! interrupt_signal)
    stop_signal_count++;
}

/* Process any pending signals. If signals are caught, this function
should be called periodically. Ideally there should never be an
unbounded amount of time when signals are not being processed.
Signal handling can restore the default colors, so callers must
immediately change colors after invoking this function. */

static void
process_signals (void)
{
while (interrupt_signal || stop_signal_count)
{
    int sig;
    int stops;
    sigset_t oldset;

    if (used_color)
        restore_default_color ();
    fflush (stdout);

    sigprocmask (SIG_BLOCK, &caught_signals, &oldset);

    /* Reload interrupt_signal and stop_signal_count, in case a new
       signal was handled before sigprocmask took effect. */
    sig = interrupt_signal;
    stops = stop_signal_count;

    /* SIGTSTP is special, since the application can receive that signal
       more than once. In this case, don't set the signal handler to the
       default. Instead, just raise the uncatchable SIGSTOP. */
    if (stops)
    {
        stop_signal_count = stops - 1;
        sig = SIGSTOP;
    }
    else
        signal (sig, SIG_DFL);

    /* Exit or suspend the program. */
    raise (sig);
    sigprocmask (SIG_SETMASK, &oldset, nullptr);

    /* If execution reaches here, then the program has been
       continued (after being suspended). */
}
}

/* Setup signal handlers if INIT is true,
otherwise restore to the default. */

static void
signal_setup (bool init)

```

```

{
/* The signals that are trapped, and the number of such signals. */
static int const sig[] =
{
    /* This one is handled specially. */
    SIGTSTP,

    /* The usual suspects. */
    SIGALRM, SIGHUP, SIGINT, SIGPIPE, SIGQUIT, SIGTERM,
#ifndef SIGPOLL
    SIGPOLL,
#endif
#ifndef SIGPROF
    SIGPROF,
#endif
#ifndef SIGVTALRM
    SIGVTALRM,
#endif
#ifndef SIGXCPU
    SIGXCPU,
#endif
#ifndef SIGXFSZ
    SIGXFSZ,
#endif
};

enum { nsigs = ARRAY_CARDINALITY (sig) };

#if ! SA_NOCLDSTOP
static bool caught_sig[nsigs];
#endif

int j;

if (init)
{
#if SA_NOCLDSTOP
    struct sigaction act;

    sigemptyset (&caught_signals);
    for (j = 0; j < nsigs; j++)
    {
        sigaction (sig[j], nullptr, &act);
        if (act.sa_handler != SIG_IGN)
            sigaddset (&caught_signals, sig[j]);
    }

    act.sa_mask = caught_signals;
    act.sa_flags = SA_RESTART;

    for (j = 0; j < nsigs; j++)
        if (sigismember (&caught_signals, sig[j]))
    {
        act.sa_handler = sig[j] == SIGTSTP ? stophandler : sighandler;
        sigaction (sig[j], &act, nullptr);
    }
#else
    for (j = 0; j < nsigs; j++)
    {
        caught_sig[j] = (signal (sig[j], SIG_IGN) != SIG_IGN);
        if (caught_sig[j])

```

```

    {
        signal (sig[j], sig[j] == SIGTSTP ? stophandler : sighandler);
        siginterrupt (sig[j], 0);
    }
#endif
}
else /* restore. */
{
#ifndef SA_NOCLDSTOP
    for (j = 0; j < nsigs; j++)
        if (sigismember (&caught_signals, sig[j]))
            signal (sig[j], SIG_DFL);
#else
    for (j = 0; j < nsigs; j++)
        if (caught_sig[j])
            signal (sig[j], SIG_DFL);
#endif
}
}

static void
signal_init (void)
{
    signal_setup (true);
}

static void
signal_restore (void)
{
    signal_setup (false);
}

int
main (int argc, char **argv)
{
    int i;
    struct pending *thispend;
    int n_files;

    initialize_main (&argc, &argv);
    set_program_name (argv[0]);
    setlocale (LC_ALL, "");
    bindtextdomain (PACKAGE, LOCALEDIR);
    textdomain (PACKAGE);

    initialize_exit_failure (LS_FAILURE);
    atexit (close_stdout);

    static_assert (ARRAY_CARDINALITY (color_indicator) + 1
                  == ARRAY_CARDINALITY (indicator_name));

    exit_status = EXIT_SUCCESS;
    print_dir_name = true;
    pending_dirs = nullptr;

    current_time.tv_sec = TYPE_MINIMUM (time_t);
    current_time.tv_nsec = -1;

    i = decode_switches (argc, argv);
}
```

```

if (print_with_color)
    parse_ls_color ();

/* Test print_with_color again, because the call to parse_ls_color
   may have just reset it -- e.g., if LS_COLORS is invalid. */

if (print_with_color)
{
    /* Don't use TAB characters in output. Some terminal
       emulators can't handle the combination of tabs and
       color codes on the same line. */
    tabsize = 0;
}

if (directories_first)
    check_symlink_mode = true;
else if (print_with_color)
{
    /* Avoid following symbolic links when possible. */
    if (is_colored (C_ORPHAN)
        || (is_colored (C_EXEC) && color_symlink_as_referent)
        || (is_colored (C_MISSING) && format == long_format))
        check_symlink_mode = true;
}

if (dereference == DEREF_UNDEFINED)
    dereference = ((immediate_dirs
                    || indicator_style == classify
                    || format == long_format)
                  ? DEREF_NEVER
                  : DEREF_COMMAND_LINE_SYMLINK_TO_DIR);

/* When using -R, initialize a data structure we'll use to
   detect any directory cycles. */
if (recursive)
{
    active_dir_set = hash_initialize (INITIAL_TABLE_SIZE, nullptr,
                                      dev_ino_hash,
                                      dev_ino_compare,
                                      dev_ino_free);
    if (active_dir_set == nullptr)
        xalloc_die ();
    obstack_init (&dev_ino_obstack);
}

localtz = tzalloc (getenv ("TZ"));

format_needs_stat = sort_type == sort_time || sort_type == sort_size
    || format == long_format
    || print_scontext
    || print_block_size;
format_needs_type = (! format_needs_stat
    && (recursive
        || print_with_color
        || indicator_style != none
        || directories_first));

if (dired)

```

```

{
obstack_init (&dired_obstack);
obstack_init (&subdired_obstack);
}

if (print_hyperlink)
{
    file_escape_init ();

    hostname = xgethostname ();
    /* The hostname is generally ignored,
       so ignore failures obtaining it. */
    if (! hostname)
        hostname = "";
}

cwd_n_alloc = 100;
cwd_file = xnmalloc (cwd_n_alloc, sizeof * cwd_file);
cwd_n_used = 0;

clear_files ();

n_files = argc - i;

if (n_files <= 0)
{
    if (immediate_dirs)
        gobble_file (".", directory, NOT_AN_INODE_NUMBER, true, "");
    else
        queue_directory (".", nullptr, true);
}
else
    do
        gobble_file (argv[i++], unknown, NOT_AN_INODE_NUMBER, true, "");
    while (i < argc);

if (cwd_n_used)
{
    sort_files ();
    if (!immediate_dirs)
        extract_dirs_from_files (nullptr, true);
    /* 'cwd_n_used' might be zero now. */
}

/* In the following if/else blocks, it is sufficient to test 'pending_dirs'
   (and not pending_dirs->name) because there may be no markers in the queue
   at this point. A marker may be enqueued when extract_dirs_from_files is
   called with a non-empty string or via print_dir. */
if (cwd_n_used)
{
    print_current_files ();
    if (pending_dirs)
        dired_outbyte ('\n');
}
else if (n_files <= 1 && pending_dirs && pending_dirs->next == 0)
    print_dir_name = false;

while (pending_dirs)
{
    thispend = pending_dirs;
}

```

```

pending_dirs = pending_dirs->next;

if (LOOP_DETECT)
{
    if (thispend->name == nullptr)
    {
        /* thispend->name == nullptr means this is a marker entry
         * indicating we've finished processing the directory.
         * Use its dev/ino numbers to remove the corresponding
         * entry from the active_dir_set hash table. */
        struct dev_ino di = dev_ino_pop ();
        struct dev_ino *found = hash_remove (active_dir_set, &di);
        if (false)
            assert_matching_dev_ino (thispend->realname, di);
        affirm (found);
        dev_ino_free (found);
        free_pending_ent (thispend);
        continue;
    }
}

print_dir (thispend->name, thispend->realname,
           thispend->command_line_arg);

free_pending_ent (thispend);
print_dir_name = true;
}

if (print_with_color && used_color)
{
    int j;

    /* Skip the restore when it would be a no-op, i.e.,
     * when left is "\033[" and right is "m". */
    if (!!(color_indicator[C_LEFT].len == 2
        && memcmp (color_indicator[C_LEFT].string, "\033[", 2) == 0
        && color_indicator[C_RIGHT].len == 1
        && color_indicator[C_RIGHT].string[0] == 'm'))
        restore_default_color ();

    fflush (stdout);

    signal_restore ();

    /* Act on any signals that arrived before the default was restored.
     * This can process signals out of order, but there doesn't seem to
     * be an easy way to do them in order, and the order isn't that
     * important anyway. */
    for (j = stop_signal_count; j; j--)
        raise (SIGSTOP);
    j = interrupt_signal;
    if (j)
        raise (j);
}

if (dired)
{
    /* No need to free these since we're about to exit. */
    dired_dump_obstack ("//DIRED//", &dired_obstack);
    dired_dump_obstack ("//SUBDIRED//", &subdired_obstack);
}

```

```

printf("//DIRED-OPTIONS// --quoting-style=%s\n",
      quoting_style_args[get_quoting_style (filename_quoting_options)]);
}

if (LOOP_DETECT)
{
    assure (hash_get_n_entries (active_dir_set) == 0);
    hash_free (active_dir_set);
}

return exit_status;
}

/* Return the line length indicated by the value given by SPEC, or -1
if unsuccessful. 0 means no limit on line length. */

static ptrdiff_t
decode_line_length (char const *spec)
{
    uintmax_t val;

    /* Treat too-large values as if they were 0, which is
       effectively infinity. */
    switch (xstrtoumax (spec, nullptr, 0, &val, ""))
    {
        case LONGINT_OK:
            return val <= MIN (PTRDIFF_MAX, SIZE_MAX) ? val : 0;

        case LONGINT_OVERFLOW:
            return 0;

        default:
            return -1;
    }
}

/* Return true if standard output is a tty, caching the result. */

static bool
stdout_isatty (void)
{
    static signed char out_tty = -1;
    if (out_tty < 0)
        out_tty = isatty (STDOUT_FILENO);
    assume (out_tty == 0 || out_tty == 1);
    return out_tty;
}

/* Set all the option flags according to the switches specified.
Return the index of the first non-option argument. */

static int
decode_switches (int argc, char **argv)
{
    char const *time_style_option = nullptr;

    /* These variables are false or -1 unless a switch says otherwise. */
    bool kibibytes_specified = false;
    int format_opt = -1;
    int hide_control_chars_opt = -1;
}

```

```

int quoting_style_opt = -1;
int sort_opt = -1;
ptrdiff_t tabsize_opt = -1;
ptrdiff_t width_opt = -1;

while (true)
{
    int oi = -1;
    int c = getopt_long (argc, argv,
                        "abcdEfgijklmnopqrstuvwxyz:xABCDEFGHILNQRST:UXZ1",
                        long_options, &oi);
    if (c == -1)
        break;

    switch (c)
    {
        case 'a':
            ignore_mode = IGNORE_MINIMAL;
            break;

        case 'b':
            quoting_style_opt = escape_quoting_style;
            break;

        case 'c':
            time_type = time_ctime;
            break;

        case 'd':
            immediate_dirs = true;
            break;

#define __MVS__
        case 'E':
        {
            int ok = do_mvs_ls(argc, argv);
            exit (ok ? EXIT_SUCCESS : EXIT_FAILURE);
        }
        break;
#define __endif

        case 'f':
            ignore_mode = IGNORE_MINIMAL; /* enable -a */
            sort_opt = sort_none; /* enable -U */
            if (format_opt == long_format)
                format_opt = -1; /* disable -l */
            print_with_color = false; /* disable --color */
            print_hyperlink = false; /* disable --hyperlink */
            print_block_size = false; /* disable -s */
            break;

        case FILE_TYPE_INDICATOR_OPTION: /* --file-type */
            indicator_style = file_type;
            break;

        case 'g':
            format_opt = long_format;
            print_owner = false;
            break;
    }
}

```

```
case 'h':
    file_human_output_opts = human_output_opts =
        human_autoscale | human_SI | human_base_1024;
    file_output_block_size = output_block_size = 1;
    break;

case 'i':
    print_inode = true;
    break;

case 'k':
    kibibytes_specified = true;
    break;

case 'l':
    format_opt = long_format;
    break;

case 'm':
    format_opt = with_commas;
    break;

case 'n':
    numeric_ids = true;
    format_opt = long_format;
    break;

case 'o': /* Just like -l, but don't display group info. */
    format_opt = long_format;
    print_group = false;
    break;

case 'p':
    indicator_style = slash;
    break;

case 'q':
    hide_control_chars_opt = true;
    break;

case 'r':
    sort_reverse = true;
    break;

case 's':
    print_block_size = true;
    break;

case 't':
    sort_opt = sort_time;
    break;

case 'u':
    time_type = time_atime;
    break;

case 'v':
    sort_opt = sort_version;
    break;
```

```

case 'w':
width_opt = decode_line_length (optarg);
if (width_opt < 0)
    error (LS_FAILURE, 0, "%s: %s", _("invalid line width"),
          quote (optarg));
break;

case 'x':
format_opt = horizontal;
break;

case 'A':
ignore_mode = IGNORE_DOT_AND_DOTDOT;
break;

case 'B':
add_ignore_pattern ("*~");
add_ignore_pattern (".*~");
break;

case 'C':
format_opt = many_per_line;
break;

case 'D':
format_opt = long_format;
print_hyperlink = false;
dired = true;
break;

case 'F':
{
    int i;
    if (optarg)
        i = XARGMATCH ("--classify", optarg, when_args, when_types);
    else
        /* Using --classify with no argument is equivalent to using
           --classify=always. */
        i = when_always;

    if (i == when_always || (i == when_if_tty && stdout_isatty ()))
        indicator_style = classify;
    break;
}

case 'G':          /* inhibit display of group info */
print_group = false;
break;

case 'H':
dereference = DEREF_COMMAND_LINE_ARGUMENTS;
break;

case DEREference_COMMAND_LINE_SYMLINK_TO_DIR_OPTION:
dereference = DEREF_COMMAND_LINE_SYMLINK_TO_DIR;
break;

case 'I':
add_ignore_pattern (optarg);
break;

```

```

case 'L':
dereference = DEREF_ALWAYS;
break;

case 'N':
quoting_style_opt = literal_quoting_style;
break;

case 'Q':
quoting_style_opt = c_quoting_style;
break;

case 'R':
recursive = true;
break;

case 'S':
sort_opt = sort_size;
break;

case 'T':
#ifndef __MVS__
{
int ok = do_mvs_ls(argc, argv);
exit (ok ? EXIT_SUCCESS : EXIT_FAILURE);
}
break;
#else
tabsize_opt = xnumtoumax (optarg, 0, 0, MIN (PTRDIFF_MAX, SIZE_MAX),
                           "", _("invalid tab size"), LS_FAILURE);
break;
#endif

case 'U':
sort_opt = sort_none;
break;

case 'X':
sort_opt = sort_extension;
break;

case '1':
/* -1 has no effect after -l. */
if (format_opt != long_format)
    format_opt = one_per_line;
break;

case AUTHOR_OPTION:
print_author = true;
break;

case HIDE_OPTION:
{
    struct ignore_pattern *hide = xmalloc (sizeof *hide);
    hide->pattern = optarg;
    hide->next = hide_patterns;
    hide_patterns = hide;
}
break;

```

```

case SORT_OPTION:
sort_opt = XARGMATCH ("--sort", optarg, sort_args, sort_types);
break;

case GROUP_DIRECTORIES_FIRST_OPTION:
directories_first = true;
break;

case TIME_OPTION:
time_type = XARGMATCH ("--time", optarg, time_args, time_types);
break;

case FORMAT_OPTION:
format_opt = XARGMATCH ("--format", optarg, format_args,
format_types);
break;

case FULL_TIME_OPTION:
format_opt = long_format;
time_style_option = "full-iso";
break;

case COLOR_OPTION:
{
    int i;
    if (optarg)
        i = XARGMATCH ("--color", optarg, when_args, when_types);
    else
        /* Using --color with no argument is equivalent to using
           --color=always. */
        i = when_always;

    print_with_color = (i == when_always
                       || (i == when_if_tty && stdout_isatty ()));
    break;
}

case HYPERLINK_OPTION:
{
    int i;
    if (optarg)
        i = XARGMATCH ("--hyperlink", optarg, when_args, when_types);
    else
        /* Using --hyperlink with no argument is equivalent to using
           --hyperlink=always. */
        i = when_always;

    print_hyperlink = (i == when_always
                      || (i == when_if_tty && stdout_isatty ()));
    break;
}

case INDICATOR_STYLE_OPTION:
indicator_style = XARGMATCH ("--indicator-style", optarg,
                             indicator_style_args,
                             indicator_style_types);
break;

case QUOTING_STYLE_OPTION:

```

```

quoting_style_opt = XARGMATCH ("--quoting-style", optarg,
                               quoting_style_args,
                               quoting_style_vals);
break;

case TIME_STYLE_OPTION:
time_style_option = optarg;
break;

case SHOW_CONTROL_CHARS_OPTION:
hide_control_chars_opt = false;
break;

case BLOCK_SIZE_OPTION:
{
    enum strtol_error e = human_options (optarg, &human_output_opts,
                                         &output_block_size);
    if (e != LONGINT_OK)
        xstrtol_fatal (e, oi, 0, long_options, optarg);
    file_human_output_opts = human_output_opts;
    file_output_block_size = output_block_size;
}
break;

case SI_OPTION:
file_human_output_opts = human_output_opts =
    human_autoscale | human_SI;
file_output_block_size = output_block_size = 1;
break;

case 'Z':
print_scontext = true;
break;

case ZERO_OPTION:
eolbyte = 0;
hide_control_chars_opt = false;
if (format_opt != long_format)
    format_opt = one_per_line;
print_with_color = false;
quoting_style_opt = literal_quoting_style;
break;

case_GETOPT_HELP_CHAR;

case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

default:
usage (LS_FAILURE);
}
}

if (!output_block_size)
{
char const *ls_block_size = getenv ("LS_BLOCK_SIZE");
human_options (ls_block_size,
               &human_output_opts, &output_block_size);
if (ls_block_size || getenv ("BLOCK_SIZE"))
{
    file_human_output_opts = human_output_opts;
}
}

```

```

        file_output_block_size = output_block_size;
    }
    if (kibibytes_specified)
    {
        human_output_opts = 0;
        output_block_size = 1024;
    }
}

format = (0 <= format_opt ? format_opt
    : ls_mode == LS_LS ? (stdout_isatty ()
        ? many_per_line : one_per_line)
    : ls_mode == LS_MULTI_COL ? many_per_line
    : /* ls_mode == LS_LONG_FORMAT */ long_format);

/* If the line length was not set by a switch but is needed to determine
   output, go to the work of obtaining it from the environment. */
ptrdiff_t linelen = width_opt;
if (format == many_per_line || format == horizontal || format == with_commas
    || print_with_color)
{
#define TIOCGWINSZ
    if (linelen < 0)
    {
        struct winsize ws;
        if (stdout_isatty ())
            && 0 <= ioctl (STDOUT_FILENO, TIOCGWINSZ, &ws)
            && 0 < ws.ws_col)
            linelen = ws.ws_col <= MIN (PTRDIFF_MAX, SIZE_MAX) ? ws.ws_col : 0;
    }
#endif
    if (linelen < 0)
    {
        char const *p = getenv ("COLUMNS");
        if (p && *p)
        {
            linelen = decode_line_length (p);
            if (linelen < 0)
                error (0, 0,
                    _("ignoring invalid width"
                    " in environment variable COLUMNS: %s"),
                    quote (p));
        }
    }
}
line_length = linelen < 0 ? 80 : linelen;

/* Determine the max possible number of display columns. */
max_idx = line_length / MIN_COLUMN_WIDTH;
/* Account for first display column not having a separator,
   or line_lengths shorter than MIN_COLUMN_WIDTH. */
max_idx += line_length % MIN_COLUMN_WIDTH != 0;

if (format == many_per_line || format == horizontal || format == with_commas)
{
    if (0 <= tabsize_opt)
        tabsize = tabsize_opt;
    else
    {

```

```

tabsize = 8;
char const *p = getenv ("TABSIZE");
if (p)
{
    uintmax_t tmp;
    if (xstroumax (p, nullptr, 0, &tmp, "") == LONGINT_OK
        && tmp <= SIZE_MAX)
        tabsize = tmp;
    else
        error (0, 0,
            _("ignoring invalid tab size"
              " in environment variable TABSIZE: %s"),
            quote (p));
}
}

qmark_funny_chars = (hide_control_chars_opt < 0
    ? ls_mode == LS_LS && stdout_isatty ()
    : hide_control_chars_opt);

int qs = quoting_style_opt;
if (qs < 0)
    qs = getenv_quoting_style ();
if (qs < 0)
    qs = (ls_mode == LS_LS
        ? (stdout_isatty () ? shell_escape_quoting_style : -1)
        : escape_quoting_style);
if (0 <= qs)
    set_quoting_style (nullptr, qs);
qs = get_quoting_style (nullptr);
align_variable_outer_quotes
    = ((format == long_format
        || ((format == many_per_line || format == horizontal) && line_length))
    && (qs == shell_quoting_style
        || qs == shell_escape_quoting_style
        || qs == c_maybe_quoting_style));
filename_quoting_options = clone_quoting_options (nullptr);
if (qs == escape_quoting_style)
    set_char_quoting (filename_quoting_options, ' ', 1);
if (file_type <= indicator_style)
{
    char const *p;
    for (p = &"*=>@|[indicator_style - file_type]; *p; p++)
        set_char_quoting (filename_quoting_options, *p, 1);
}

dirname_quoting_options = clone_quoting_options (nullptr);
set_char_quoting (dirname_quoting_options, ':', 1);

/* --dired implies --format=long (-l) and sans --hyperlink.
   So ignore it if those overridden. */
dired &= (format == long_format) & !print_hyperlink;

if (eolbyte < dired)
    error (LS_FAILURE, 0, _("--dired and --zero are incompatible"));

/* If -c or -u is specified and not -l (or any other option that implies -l),
   and no sort-type was specified, then sort by the ctime (-c) or atime (-u).
   The behavior of ls when using either -c or -u but with neither -l nor -t

```

appears to be unspecified by POSIX. So, with GNU ls, '-u' alone means sort by atime (this is the one that's not specified by the POSIX spec), -lu means show atime and sort by name, -lut means show atime and sort by atime. \*/

```
sort_type = (0 <= sort_opt ? sort_opt
  : (format != long_format
      && (time_type == time_ctime || time_type == time_atime
           || time_type == time_btime))
  ? sort_time : sort_name);

if (format == long_format)
{
  char const *style = time_style_option;
  static char const posix_prefix[] = "posix-";

  if (! style)
  {
    style = getenv ("TIME_STYLE");
    if (! style)
      style = "locale";
  }

while (STREQ_LEN (style, posix_prefix, sizeof posix_prefix - 1))
{
  if (! hard_locale (LC_TIME))
    return optind;
  style += sizeof posix_prefix - 1;
}

if (*style == '+')
{
  char const *p0 = style + 1;
  char *p0nl = strchr (p0, '\n');
  char const *p1 = p0;
  if (p0nl)
  {
    if (strchr (p0nl + 1, '\n'))
      error (LS_FAILURE, 0, _("invalid time style format %s"),
             quote (p0));
    *p0nl++ = '\0';
    p1 = p0nl;
  }
  long_time_format[0] = p0;
  long_time_format[1] = p1;
}
else
{
  ptrdiff_t res = argmatch (style, time_style_args,
                            (char const *) time_style_types,
                            sizeof (*time_style_types));
  if (res < 0)
  {
    /* This whole block used to be a simple use of XARGMATCH.
       but that didn't print the "posix-"-prefixed variants or
       the "+"-prefixed format string option upon failure. */
    argmatch_invalid ("time style", style, res);

    /* The following is a manual expansion of argmatch_valid,
       but with the added "+ ..." description and the [posix-]
```

```

prefixes prepended. Note that this simplification works
only because all four existing time_style_types values
are distinct. */
fputs (_("Valid arguments are:\n"), stderr);
char const *const *p = time_style_args;
while (*p)
    fprintf (stderr, " - [posix-]%s\n", *p++);
fputs (_(" - +FORMAT (e.g., +%H:%M) for a 'date'-style"
        " format\n"), stderr);
usage (LS_FAILURE);
}
switch (res)
{
case full_iso_time_style:
long_time_format[0] = long_time_format[1] =
    "%Y-%m-%d %H:%M:%S.%N %z";
break;

case long_iso_time_style:
long_time_format[0] = long_time_format[1] = "%Y-%m-%d %H:%M";
break;

case iso_time_style:
long_time_format[0] = "%Y-%m-%d ";
long_time_format[1] = "%m-%d %H:%M";
break;

case locale_time_style:
if (hard_locale (LC_TIME))
{
    for (int i = 0; i < 2; i++)
        long_time_format[i] =
            dcgettext (nullptr, long_time_format[i], LC_TIME);
}
}

abformat_init ();
}

return optind;
}

/* Parse a string as part of the LS_COLORS variable; this may involve
decoding all kinds of escape characters. If equals_end is set an
unescape equal sign ends the string, otherwise only a : or \0
does. Set *OUTPUT_COUNT to the number of bytes output. Return
true if successful.

```

The resulting string is *\*not\** null-terminated, but may contain embedded nulls.

Note that both dest and src are char \*\*; on return they point to the first free byte after the array and the character that ended the input string, respectively. \*/

```

static bool
get_funky_string (char **dest, char const **src, bool equals_end,
                  size_t *output_count)
{

```

```

char num;                      /* For numerical codes */
size_t count;                  /* Something to count with */
enum {
    ST_GND, ST_BACKSLASH, ST_OCTAL, ST_HEX, ST_CARET, ST_END, ST_ERROR
} state;
char const *p;
char *q;

p = *src;                      /* We don't want to double-indirect */
q = *dest;                     /* the whole darn time. */

count = 0;                      /* No characters counted in yet. */
num = 0;

state = ST_GND;                /* Start in ground state. */
while (state < ST_END)
{
    switch (state)
    {
        case ST_GND:          /* Ground state (no escapes) */
        switch (*p)
        {
            case '\0':
            case '\\':
                state = ST_BACKSLASH; /* Backslash escape sequence */
                ++p;
                break;
            case '^':
                state = ST_CARET; /* Caret escape */
                ++p;
                break;
            case '=':
                if (equals_end)
                {
                    state = ST_END; /* End */
                    break;
                }
                FALLTHROUGH;
            default:
                *(q++) = *(p++);
                ++count;
                break;
        }
        break;

        case ST_BACKSLASH: /* Backslash escaped character */
        switch (*p)
        {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
                state = ST_OCTAL; /* Octal sequence */

```

```

num = *p - '0';
break;
case 'x':
case 'X':
state = ST_HEX;      /* Hex sequence */
num = 0;
break;
case 'a':           /* Bell */
num = '\a';
break;
case 'b':           /* Backspace */
num = '\b';
break;
case 'e':           /* Escape */
num = 27;
break;
case 'f':           /* Form feed */
num = '\f';
break;
case 'n':           /* Newline */
num = '\n';
break;
case 'r':           /* Carriage return */
num = '\r';
break;
case 't':           /* Tab */
num = '\t';
break;
case 'v':           /* Vtab */
num = '\v';
break;
case '?':           /* Delete */
num = 127;
break;
case '_':           /* Space */
num = ' ';
break;
case '0':           /* End of string */
state = ST_ERROR; /* Error! */
break;
default:            /* Escaped character like \ ^ : = */
num = *p;
break;
}
if (state == ST_BACKSLASH)
{
*(q++) = num;
++count;
state = ST_GND;
}
++p;
break;

case ST_OCTAL:          /* Octal sequence */
if (*p < '0' || *p > '7')
{
*(q++) = num;
++count;
state = ST_GND;
}

```

```

else
    num = (num << 3) + (*p++ - '0');
break;

case ST_HEX:           /* Hex sequence */
switch (*p)
{
    case '0':
    case '1':
    case '2':
    case '3':
    case '4':
    case '5':
    case '6':
    case '7':
    case '8':
    case '9':
        num = (num << 4) + (*p++ - '0');
        break;
    case 'a':
    case 'b':
    case 'c':
    case 'd':
    case 'e':
    case 'f':
        num = (num << 4) + (*p++ - 'a') + 10;
        break;
    case 'A':
    case 'B':
    case 'C':
    case 'D':
    case 'E':
    case 'F':
        num = (num << 4) + (*p++ - 'A') + 10;
        break;
    default:
        *(q++) = num;
        ++count;
        state = ST_GND;
        break;
}
break;

case ST_CARET:         /* Caret escape */
state = ST_GND;        /* Should be the next state... */
if (*p >= '@' && *p <= '~')
{
    *(q++) = *(p++) & 037;
    ++count;
}
else if (*p == '?')
{
    *(q++) = 127;
    ++count;
}
else
    state = ST_ERROR;
break;

default:

```

```

        unreachable ();
    }

}

*dest = q;
*src = p;
*output_count = count;

return state != ST_ERROR;
}

enum parse_state
{
    PS_START = 1,
    PS_2,
    PS_3,
    PS_4,
    PS_DONE,
    PS_FAIL
};

/* Check if the content of TERM is a valid name in dircolors. */

static bool
known_term_type (void)
{
char const *term = getenv ("TERM");
if (! term || ! *term)
    return false;

char const *line = G_line;
while (line - G_line < sizeof (G_line))
{
    if (STRNCMP_LIT (line, "TERM ") == 0)
    {
        if (fnmatch (line + 5, term, 0) == 0)
            return true;
    }
    line += strlen (line) + 1;
}

return false;
}

static void
parse_ls_color (void)
{
char const *p;          /* Pointer to character being parsed */
char *buf;              /* color_buf buffer pointer */
int ind_no;             /* Indicator number */
char label[3];          /* Indicator label */
struct color_ext_type *ext; /* Extension we are working on */

if ((p = getenv ("LS_COLORS")) == nullptr || *p == '\0')
{
    /* LS_COLORS takes precedence, but if that's not set then
       honor the COLORTERM and TERM env variables so that
       we only go with the internal ANSI color codes if the
       former is non empty or the latter is set to a known value. */
}

```

```

char const *colorterm = getenv ("COLORTERM");
if (! (colorterm && *colorterm) && ! known_term_type ())
    print_with_color = false;
return;
}

ext = nullptr;
strcpy (label, "??");

/* This is an overly conservative estimate, but any possible
   LS_COLORS string will *not* generate a color_buf longer than
   itself, so it is a safe way of allocating a buffer in
   advance. */
buf = color_buf = xstrdup (p);

enum parse_state state = PS_START;
while (true)
{
    switch (state)
    {
        case PS_START:           /* First label character */
            switch (*p)
            {
                case ':':
                    ++p;
                    break;

                case '*':
                    /* Allocate new extension block and add to head of
                       linked list (this way a later definition will
                       override an earlier one, which can be useful for
                       having terminal-specific defs override global). */

                    ext = xmalloc (sizeof *ext);
                    ext->next = color_ext_list;
                    color_ext_list = ext;
                    ext->exact_match = false;

                    ++p;
                    ext->ext.string = buf;

                    state = (get_funky_string (&buf, &p, true, &ext->ext.len)
                           ? PS_4 : PS_FAIL);
                    break;

                case '\0':
                    state = PS_DONE;    /* Done! */
                    goto done;

                default:              /* Assume it is file type label */
                    label[0] = *(p++);
                    state = PS_2;
                    break;
            }
            break;

        case PS_2:               /* Second label character */
            if (*p)
            {
                label[1] = *(p++);

```

```

        state = PS_3;
    }
else
    state = PS_FAIL; /* Error */
break;

case PS_3:           /* Equal sign after indicator label */
state = PS_FAIL; /* Assume failure... */
if (*(p++) == '=')/* It *should* be... */
{
    for (ind_no = 0; indicator_name[ind_no] != nullptr; ++ind_no)
    {
        if (STREQ (label, indicator_name[ind_no]))
        {
            color_indicator[ind_no].string = buf;
            state = (get_funky_string (&buf, &p, false,
                                         &color_indicator[ind_no].len)
                      ? PS_START : PS_FAIL);
            break;
        }
    }
    if (state == PS_FAIL)
        error (0, 0, _("unrecognized prefix: %s"), quote (label));
}
break;

case PS_4:           /* Equal sign after *.ext */
if (*(p++) == '=')
{
    ext->seq.string = buf;
    state = (get_funky_string (&buf, &p, false, &ext->seq.len)
              ? PS_START : PS_FAIL);
}
else
    state = PS_FAIL;
break;

case PS_FAIL:
goto done;

default:
affirm (false);
}
}
done:

if (state == PS_FAIL)
{
struct color_ext_type *e;
struct color_ext_type *e2;

error (0, 0,
       _("unparseable value for LS_COLORS environment variable"));
free (color_buf);
for (e = color_ext_list; e != nullptr; /* empty */)
{
    e2 = e;
    e = e->next;
    free (e2);
}
}

```

```

        print_with_color = false;
    }
else
{
/* Postprocess list to set EXACT_MATCH on entries where there are
   different cased extensions with separate sequences defined.
   Also set ext.len to SIZE_MAX on any entries that can't
   match due to precedence, to avoid redundant string compares. */
struct color_ext_type *e1;

for (e1 = color_ext_list; e1 != nullptr; e1 = e1->next)
{
    struct color_ext_type *e2;
    bool case_ignored = false;

    for (e2 = e1->next; e2 != nullptr; e2 = e2->next)
    {
        if (e2->ext.len < SIZE_MAX && e1->ext.len == e2->ext.len)
        {
            if (memcmp (e1->ext.string, e2->ext.string, e1->ext.len) == 0)
                e2->ext.len = SIZE_MAX; /* Ignore */
            else if (c_strncasecmp (e1->ext.string, e2->ext.string,
                                   e1->ext.len) == 0)
            {
                if (case_ignored)
                {
                    e2->ext.len = SIZE_MAX; /* Ignore */
                }
                else if (e1->seq.len == e2->seq.len
                         && memcmp (e1->seq.string, e2->seq.string,
                                     e1->seq.len) == 0)
                {
                    e2->ext.len = SIZE_MAX; /* Ignore */
                    case_ignored = true; /* Ignore all subsequent */
                }
                else
                {
                    e1->exact_match = true;
                    e2->exact_match = true;
                }
            }
        }
    }
}

if (color_indicator[C_LINK].len == 6
    && !STRNCMP_LIT (color_indicator[C_LINK].string, "target"))
    color_symlink_as_referent = true;
}

/* Return the quoting style specified by the environment variable
QUOTING_STYLE if set and valid, -1 otherwise. */

static int
getenv_quoting_style (void)
{
char const *q_style = getenv ("QUOTING_STYLE");
if (!q_style)
    return -1;
}

```

```

int i = ARGMATCH (q_style, quoting_style_args, quoting_style_vals);
if (i < 0)
{
    error (0, 0,
        ("ignoring invalid value"
         " of environment variable QUOTING_STYLE: %s"),
        quote (q_style));
    return -1;
}
return quoting_style_vals[i];
}

/* Set the exit status to report a failure. If SERIOUS, it is a
serious failure; otherwise, it is merely a minor problem. */

static void
set_exit_status (bool serious)
{
if (serious)
    exit_status = LS_FAILURE;
else if (exit_status == EXIT_SUCCESS)
    exit_status = LS_MINOR_PROBLEM;
}

/* Assuming a failure is serious if SERIOUS, use the printf-style
MESSAGE to report the failure to access a file named FILE. Assume
errno is set appropriately for the failure. */

static void
file_failure (bool serious, char const *message, char const *file)
{
error (0, errno, message, quoteaf (file));
set_exit_status (serious);
}

/* Request that the directory named NAME have its contents listed later.
If REALNAME is nonzero, it will be used instead of NAME when the
directory name is printed. This allows symbolic links to directories
to be treated as regular directories but still be listed under their
real names. NAME == nullptr is used to insert a marker entry for the
directory named in REALNAME.
If NAME is non-null, we use its dev/ino information to save
a call to stat -- when doing a recursive (-R) traversal.
COMMAND_LINE_ARG means this directory was mentioned on the command line. */

static void
queue_directory (char const *name, char const *realname, bool command_line_arg)
{
struct pending *new = xmalloc (sizeof *new);
new->realname = realname ? xstrdup (realname) : nullptr;
new->name = name ? xstrdup (name) : nullptr;
new->command_line_arg = command_line_arg;
new->next = pending_dirs;
pending_dirs = new;
}

/* Read directory NAME, and list the files in it.
If REALNAME is nonzero, print its name instead of NAME;
this is used for symbolic links to directories.
COMMAND_LINE_ARG means this directory was mentioned on the command line. */

```

```

static void
print_dir (char const *name, char const *realname, bool command_line_arg)
{
DIR *dirp;
struct dirent *next;
uintmax_t total_blocks = 0;
static bool first = true;

errno = 0;
dirp = opendir (name);
if (!dirp)
{
    file_failure (command_line_arg, _("cannot open directory %s"), name);
    return;
}

if (LOOP_DETECT)
{
    struct stat dir_stat;
    int fd = dirfd (dirp);

/* If dirfd failed, endure the overhead of stat'ing by path */
if ((0 <= fd
    ? fstat_for_ino (fd, &dir_stat)
    : stat_for_ino (name, &dir_stat)) < 0)
{
    file_failure (command_line_arg,
        _("cannot determine device and inode of %s"), name);
    closedir (dirp);
    return;
}

/* If we've already visited this dev/inode pair, warn that
   we've found a loop, and do not process this directory. */
if (visit_dir (dir_stat.st_dev, dir_stat.st_ino))
{
    error (0, 0, _("%s: not listing already-listed directory"),
        quotef (name));
    closedir (dirp);
    set_exit_status (true);
    return;
}

dev_ino_push (dir_stat.st_dev, dir_stat.st_ino);
}

clear_files ();

if (recursive || print_dir_name)
{
    if (!first)
        dired_outbyte ('\n');
    first = false;
    dired_indent ();

    char *absolute_name = nullptr;
    if (print_hyperlink)
    {
        absolute_name = canonicalize_filename_mode (name, CAN_MISSING);
}

```

```

    if (! absolute_name)
        file_failure (command_line_arg,
                      _("error canonicalizing %s"), name);
    }
    quote_name (realname ? realname : name, dirname_quoting_options, -1,
                nullptr, true, &subdired_obstack, absolute_name);

    free (absolute_name);

    dired_outstring (" :\n");
}

/* Read the directory entries, and insert the subfiles into the 'cwd_file'
table. */

while (true)
{
    /* Set errno to zero so we can distinguish between a readdir failure
       and when readdir simply finds that there are no more entries. */
    errno = 0;
    next = readdir (dirp);
    /* Some readdir()s do not absorb ENOENT (dir deleted but open). */
    if (errno == ENOENT)
        errno = 0;
    if (next)
    {
        if (! file_ignored (next->d_name))
        {
            enum filetype type = unknown;

#if HAVE_STRUCT_DIRENT_D_TYPE
            switch (next->d_type)
            {
                case DT_BLK: type = blockdev; break;
                case DT_CHR: type = chardev; break;
                case DT_DIR: type = directory; break;
                case DT_FIFO: type = fifo; break;
                case DT_LNK: type = symbolic_link; break;
                case DT_REG: type = normal; break;
                case DT_SOCK: type = sock; break;
#endif
                #ifdef DT_WHT
                case DT_WHT: type = whiteout; break;
#endif
            }
#endif
            total_blocks += gobble_file (next->d_name, type,
                                         RELIABLE_D_INO (next),
                                         false, name);

            /* In this narrow case, print out each name right away, so
               ls uses constant memory while processing the entries of
               this directory. Useful when there are many (millions)
               of entries in a directory. */
            if (format == one_per_line && sort_type == sort_none
                && !print_block_size && !recursive)
            {
                /* We must call sort_files in spite of
                   "sort_type == sort_none" for its initialization
                   of the sorted_file vector. */
                sort_files ();
            }
        }
    }
}

```

```

        print_current_files ();
        clear_files ();
    }
}

else if (errno != 0)
{
    file_failure (command_line_arg, _("reading directory %s"), name);
    if (errno != EOVERRFLOW)
        break;
}
else
    break;

/* When processing a very large directory, and since we've inhibited
   interrupts, this loop would take so long that ls would be annoyingly
   uninterruptible. This ensures that it handles signals promptly. */
process_signals ();
}

if (closedir (dirp) != 0)
{
    file_failure (command_line_arg, _("closing directory %s"), name);
    /* Don't return; print whatever we got. */
}

/* Sort the directory contents. */
sort_files ();

/* If any member files are subdirectories, perhaps they should have their
   contents listed rather than being mentioned here as files. */

if (recursive)
    extract_dirs_from_files (name, false);

if (format == long_format || print_block_size)
{
    char buf[LONGEST_HUMAN_READABLE + 3];
    char *p = human_readable (total_blocks, buf + 1, human_output_opts,
                             ST_NBLOCKSIZE, output_block_size);
    char *pend = p + strlen (p);
    *--p = ' ';
    *pend++ = eolbyte;
    dirent_indent ();
    dirent_outstring (_("total"));
    dirent_outbuf (p, pend - p);
}

if (cwd_n_used)
    print_current_files ();
}

/* Add 'pattern' to the list of patterns for which files that match are
not listed. */

static void
add_ignore_pattern (char const *pattern)
{
    struct ignore_pattern *ignore;

```

```

ignore = xmalloc (sizeof *ignore);
ignore->pattern = pattern;
/* Add it to the head of the linked list. */
ignore->next = ignore_patterns;
ignore_patterns = ignore;
}

/* Return true if one of the PATTERNS matches FILE. */

static bool
patterns_match (struct ignore_pattern const *patterns, char const *file)
{
struct ignore_pattern const *p;
for (p = patterns; p; p = p->next)
    if (fnmatch (p->pattern, file, FNM_PERIOD) == 0)
        return true;
return false;
}

/* Return true if FILE should be ignored. */

static bool
file_ignored (char const *name)
{
return ((ignore_mode != IGNORE_MINIMAL
    && name[0] == '.'
    && (ignore_mode == IGNORE_DEFAULT || ! name[1 + (name[1] == '.')]))
    || (ignore_mode == IGNORE_DEFAULT
        && patterns_match (hide_patterns, name))
    || patterns_match (ignore_patterns, name));
}

/* POSIX requires that a file size be printed without a sign, even
when negative. Assume the typical case where negative sizes are
actually positive values that have wrapped around. */

static uintmax_t
unsigned_file_size (off_t size)
{
return size + (size < 0) * ((uintmax_t) OFF_T_MAX - OFF_T_MIN + 1);
}

#ifndef HAVE_CAP
/* Return true if NAME has a capability (see linux/capability.h) */
static bool
has_capability (char const *name)
{
char *result;
bool has_cap;

cap_t cap_d = cap_get_file (name);
if (cap_d == nullptr)
    return false;

result = cap_to_text (cap_d, nullptr);
cap_free (cap_d);
if (!result)
    return false;
}

/* check if human-readable capability string is empty */

```

```

has_cap = !!*result;

cap_free (result);
return has_cap;
}
#else
static bool
has_capability (MAYBE_UNUSED char const *name)
{
errno = ENOTSUP;
return false;
}
#endif

/* Enter and remove entries in the table 'cwd_file'. */

static void
free_ent (struct fileinfo *f)
{
free (f->name);
free (f->linkname);
free (f->absolute_name);
if (f->scontext != UNKNOWN_SECURITY_CONTEXT)
{
if (is_smack_enabled ())
    free (f->scontext);
else
    freecon (f->scontext);
}
}

/* Empty the table of files. */
static void
clear_files (void)
{
for (size_t i = 0; i < cwd_n_used; i++)
{
    struct fileinfo *f = sorted_file[i];
    free_ent (f);
}

cwd_n_used = 0;
cwd_some_quoted = false;
any_has_acl = false;
inode_number_width = 0;
block_size_width = 0;
nlink_width = 0;
owner_width = 0;
group_width = 0;
author_width = 0;
scontext_width = 0;
major_device_number_width = 0;
minor_device_number_width = 0;
file_size_width = 0;
}

/* Return true if ERR implies lack-of-support failure by a
getxattr-calling function like getfilecon or file_has_acl. */
static bool
errno_unsupported (int err)

```

```

{
return (err == EINVAL || err == ENOSYS || is_ENOTSUP (err));
}

/* Cache *getfilecon failure, when it's trivial to do so.
Like getfilecon/lgetfilecon, but when F's st_dev says it's doesn't
support getting the security context, fail with ENOTSUP immediately. */
static int
getfilecon_cache (char const *file, struct fileinfo *f, bool deref)
{
/* st_dev of the most recently processed device for which we've
   found that []getfilecon fails indicating lack of support. */
static dev_t unsupported_device;

if (f->stat.st_dev == unsupported_device)
{
    errno = ENOTSUP;
    return -1;
}
int r = 0;
#ifndef HAVE_SMACK
if (is_smack_enabled ())
    r = smack_new_label_from_path (file, "security.SMACK64", deref,
                                  &f->scontext);
#else
r = (deref
      ? getfilecon (file, &f->scontext)
      : lgetfilecon (file, &f->scontext));
#endif
if (r < 0 && errno_unsupported (errno))
    unsupported_device = f->stat.st_dev;
return r;
}

/* Cache file_has_acl failure, when it's trivial to do.
Like file_has_acl, but when F's st_dev says it's on a file
system lacking ACL support, return 0 with ENOTSUP immediately. */
static int
file_has_acl_cache (char const *file, struct fileinfo *f)
{
/* st_dev of the most recently processed device for which we've
   found that file_has_acl fails indicating lack of support. */
static dev_t unsupported_device;

if (f->stat.st_dev == unsupported_device)
{
    errno = ENOTSUP;
    return 0;
}

/* Zero errno so that we can distinguish between two 0-returning cases:
   "has-ACL-support, but only a default ACL" and "no ACL support". */
errno = 0;
int n = file_has_acl (file, &f->stat);
if (n <= 0 && errno_unsupported (errno))
    unsupported_device = f->stat.st_dev;
return n;
}

/* Cache has_capability failure, when it's trivial to do.

```

Like has\_capability, but when F's st\_dev says it's on a file system lacking capability support, return 0 with ENOTSUP immediately. \*/

```

static bool
has_capability_cache (char const *file, struct fileinfo *f)
{
    /* st_dev of the most recently processed device for which we've
       found that has_capability fails indicating lack of support. */
    static dev_t unsupported_device;

    if (f->stat.st_dev == unsupported_device)
    {
        errno = ENOTSUP;
        return 0;
    }

    bool b = has_capability (file);
    if ( !b && errno_unsupported (errno))
        unsupported_device = f->stat.st_dev;
    return b;
}

static bool
needs_quoting (char const *name)
{
    char test[2];
    size_t len = quotearg_buffer (test, sizeof test , name, -1,
                                  filename_quoting_options);
    return *name != *test || strlen (name) != len;
}

/* Add a file to the current table of files.
   Verify that the file exists, and print an error message if it does not.
   Return the number of blocks that the file occupies. */
static uintmax_t
gobble_file (char const *name, enum filetype type, ino_t inode,
             bool command_line_arg, char const *dirname)
{
    uintmax_t blocks = 0;
    struct fileinfo *f;

    /* An inode value prior to gobble_file necessarily came from readdir,
       which is not used for command line arguments. */
    affirm (! command_line_arg || inode == NOT_AN_INODE_NUMBER);

    if (cwd_n_used == cwd_n_alloc)
    {
        cwd_file = xnrealloc (cwd_file, cwd_n_alloc, 2 * sizeof *cwd_file);
        cwd_n_alloc *= 2;
    }

    f = &cwd_file[cwd_n_used];
    memset (f, '\0', sizeof *f);
    f->stat.st_ino = inode;
    f->filetype = type;

    f->quoted = -1;
    if (! cwd_some_quoted) && align_variable_outer_quotes)
    {
        /* Determine if any quoted for padding purposes. */
        f->quoted = needs_quoting (name);
    }
}

```



```

if (!f->absolute_name)
    file_failure (command_line_arg,
                  _("error canonicalizing %s"), full_name);
}

switch (dereference)
{
case DEREF_ALWAYS:
err = do_stat (full_name, &f->stat);
do_deref = true;
break;

case DEREF_COMMAND_LINE_ARGUMENTS:
case DEREF_COMMAND_LINE_SYMLINK_TO_DIR:
if (command_line_arg)
{
    bool need_lstat;
    err = do_stat (full_name, &f->stat);
    do_deref = true;

if (dereference == DEREF_COMMAND_LINE_ARGUMENTS)
    break;

need_lstat = (err < 0
              ? (errno == ENOENT || errno == ELOOP)
              : ! S_ISDIR (f->stat.st_mode));
if (!need_lstat)
    break;

/* stat failed because of ENOENT || ELOOP, maybe indicating a
   non-traversable symlink. Or stat succeeded,
   FULL_NAME does not refer to a directory,
   and --dereference-command-line-symlink-to-dir is in effect.
   Fall through so that we call lstat instead. */
}
FALLTHROUGH;

default: /* DEREF_NEVER */
err = do_lstat (full_name, &f->stat);
do_deref = false;
break;
}

if (err != 0)
{
/* Failure to stat a command line argument leads to
   an exit status of 2. For other files, stat failure
   provokes an exit status of 1. */
file_failure (command_line_arg,
                  _("cannot access %s"), full_name);

f->scontext = UNKNOWN_SECURITY_CONTEXT;

if (command_line_arg)
    return 0;

f->name = xstrdup (name);
cwd_n_used++;

return 0;

```

```

    }

f->stat_ok = true;

/* Note has_capability() adds around 30% runtime to 'ls --color' */
if ((type == normal || S_ISREG (f->stat.st_mode))
    && print_with_color && is_colored (C_CAP))
    f->has_capability = has_capability_cache (full_name, f);

if (format == long_format || print_scontext)
{
    bool have_scontext = false;
    bool have_acl = false;
    int attr_len = getfilecon_cache (full_name, f, do_deref);
    err = (attr_len < 0);

    if (err == 0)
    {
        if (is_smack_enabled ())
            have_scontext = ! STREQ ("_", f->scontext);
        else
            have_scontext = ! STREQ ("unlabeled", f->scontext);
    }
    else
    {
        f->scontext = UNKNOWN_SECURITY_CONTEXT;
    }
}

/* When requesting security context information, don't make
   ls fail just because the file (even a command line argument)
   isn't on the right type of file system. I.e., a getfilecon
   failure isn't in the same class as a stat failure. */
if (is_ENOTSUP (errno) || errno == ENODATA)
    err = 0;
}

if (err == 0 && format == long_format)
{
    int n = file_has_acl_cache (full_name, f);
    err = (n < 0);
    have_acl = (0 < n);
}

f->acl_type = (!have_scontext && !have_acl
    ? ACL_T_NONE
    : (have_scontext && !have_acl
        ? ACL_T_LSM_CONTEXT_ONLY
        : ACL_T_YES));
any_has_acl |= f->acl_type != ACL_T_NONE;

if (err)
    error (0, errno, "%s", quotef (full_name));
}

if (S_ISLNK (f->stat.st_mode)
    && (format == long_format || check_symlink_mode))
{
    struct stat linkstats;

    get_link_name (full_name, f, command_line_arg);
}

```

```

/* Use the slower quoting path for this entry, though
   don't update CWD_SOME_QUOTED since alignment not affected. */
if (f->linkname && f->quoted == 0 && needs_quoting (f->linkname))
    f->quoted = -1;

/* Avoid following symbolic links when possible, i.e., when
   they won't be traced and when no indicator is needed. */
if (f->linkname
    && (file_type <= indicator_style || check_symlink_mode)
    && stat_for_mode (full_name, &linkstats) == 0)
{
    f->linkok = true;
    f->linkmode = linkstats.st_mode;
}
}

if (S_ISLNK (f->stat.st_mode))
    f->filetype = symbolic_link;
else if (S_ISDIR (f->stat.st_mode))
{
    if (command_line_arg && !immediate_dirs)
        f->filetype = arg_directory;
    else
        f->filetype = directory;
}
else
    f->filetype = normal;

blocks = STP_NBLOCKS (&f->stat);
if (format == long_format || print_block_size)
{
    char buf[LONGEST_HUMAN_READABLE + 1];
    int len = mbswidth (human_readable (blocks, buf, human_output_opts,
                                       ST_NBLOCKSIZE, output_block_size),
                         MBSWIDTH_FLAGS);
    if (block_size_width < len)
        block_size_width = len;
}

if (format == long_format)
{
    if (print_owner)
    {
        int len = format_user_width (f->stat.st_uid);
        if (owner_width < len)
            owner_width = len;
    }

    if (print_group)
    {
        int len = format_group_width (f->stat.st_gid);
        if (group_width < len)
            group_width = len;
    }

    if (print_author)
    {
        int len = format_user_width (f->stat.st_author);
        if (author_width < len)
            author_width = len;
    }
}

```

```

        }

    }

if (print_scontext)
{
    int len = strlen (f->scontext);
    if (scontext_width < len)
        scontext_width = len;
}

if (format == long_format)
{
    char b[INT_BUFSIZE_BOUND (uintmax_t)];
    int b_len = strlen (umaxtostr (f->stat.st_nlink, b));
    if (nlink_width < b_len)
        nlink_width = b_len;

if (S_ISCHR (f->stat.st_mode) || S_ISBLK (f->stat.st_mode))
{
    char buf[INT_BUFSIZE_BOUND (uintmax_t)];
    int len = strlen (umaxtostr (major (f->stat.st_rdev), buf));
    if (major_device_number_width < len)
        major_device_number_width = len;
    len = strlen (umaxtostr (minor (f->stat.st_rdev), buf));
    if (minor_device_number_width < len)
        minor_device_number_width = len;
    len = major_device_number_width + 2 + minor_device_number_width;
    if (file_size_width < len)
        file_size_width = len;
}
else
{
    char buf[LONGEST_HUMAN_READABLE + 1];
    uintmax_t size = unsigned_file_size (f->stat.st_size);
    int len = mbswidth (human_readable (size, buf,
                                         file_human_output_opts,
                                         1, file_output_block_size),
                         MBSWIDTH_FLAGS);
    if (file_size_width < len)
        file_size_width = len;
}
}

if (print_inode)
{
    char buf[INT_BUFSIZE_BOUND (uintmax_t)];
    int len = strlen (umaxtostr (f->stat.st_ino, buf));
    if (inode_number_width < len)
        inode_number_width = len;
}

f->name = xstrdup (name);
cwd_n_used++;

return blocks;
}

/* Return true if F refers to a directory. */
static bool
```

```

is_directory (const struct fileinfo *f)
{
    return f->filetype == directory || f->filetype == arg_directory;
}

/* Return true if F refers to a (symlinked) directory. */
static bool
is_linked_directory (const struct fileinfo *f)
{
    return f->filetype == directory || f->filetype == arg_directory
        || S_ISDIR (f->linkmode);
}

/* Put the name of the file that FILENAME is a symbolic link to
into the LINKNAME field of 'f'. COMMAND_LINE_ARG indicates whether
FILENAME is a command-line argument. */

static void
get_link_name (char const *filename, struct fileinfo *f, bool command_line_arg)
{
    f->linkname = areadlink_with_size (filename, f->stat.st_size);
    if (f->linkname == nullptr)
        file_failure (command_line_arg, _("cannot read symbolic link %s"),
                      filename);
}

/* Return true if the last component of NAME is '.' or '..'
This is so we don't try to recurse on './././.. .' */

static bool
basename_is_dot_or_dotdot (char const *name)
{
    char const *base = last_component (name);
    return dot_or_dotdot (base);
}

/* Remove any entries from CWD_FILE that are for directories,
and queue them to be listed as directories instead.
DIRNAME is the prefix to prepend to each dirname
to make it correct relative to ls's working dir;
if it is null, no prefix is needed and "." and ".." should not be ignored.
If COMMAND_LINE_ARG is true, this directory was mentioned at the top level,
This is desirable when processing directories recursively. */

static void
extract_dirs_from_files (char const *dirname, bool command_line_arg)
{
    size_t i;
    size_t j;
    bool ignore_dot_and_dot_dot = (dirname != nullptr);

    if (dirname && LOOP_DETECT)
    {
        /* Insert a marker entry first. When we dequeue this marker entry,
           we'll know that DIRNAME has been processed and may be removed
           from the set of active directories. */
        queue_directory (nullptr, dirname, false);
    }

    /* Queue the directories last one first, because queueing reverses the

```

```

    order. */
for (i = cwd_n_used; i-- != 0; )
{
    struct fileinfo *f = sorted_file[i];

    if (is_directory (f)
        && (! ignore_dot_and_dot_dot
            || ! basename_is_dot_or_dotdot (f->name)))
    {
        if (!dirname || f->name[0] == '/')
            queue_directory (f->name, f->linkname, command_line_arg);
        else
        {
            char *name = file_name_concat (dirname, f->name, nullptr);
            queue_directory (name, f->linkname, command_line_arg);
            free (name);
        }
        if (f->filetype == arg_directory)
            free_ent (f);
    }
}

/* Now delete the directories from the table, compacting all the remaining
   entries. */

for (i = 0, j = 0; i < cwd_n_used; i++)
{
    struct fileinfo *f = sorted_file[i];
    sorted_file[j] = f;
    j += (f->filetype != arg_directory);
}
cwd_n_used = j;

/* Use strcoll to compare strings in this locale. If an error occurs,
   report an error and longjmp to failed_strcoll. */

static jmp_buf failed_strcoll;

static int
xstrcoll (char const *a, char const *b)
{
    int diff;
    errno = 0;
    diff = strcoll (a, b);
    if (errno)
    {
        error (0, errno, _("cannot compare file names %s and %s"),
               quote_n (0, a), quote_n (1, b));
        set_exit_status (false);
        longjmp (failed_strcoll, 1);
    }
    return diff;
}

/* Comparison routines for sorting the files. */

typedef void const *V;
typedef int (*qsortFunc)(V a, V b);

```

```

/* Used below in DEFINE_SORT_FUNCTIONS for _df_ sort function variants. */
static int
dirfirst_check (struct fileinfo const *a, struct fileinfo const *b,
                int (*cmp) (V, V))
{
int diff = is_linked_directory (b) - is_linked_directory (a);
return diff ? diff : cmp (a, b);
}

/* Define the 8 different sort function variants required for each sortkey.
KEY_NAME is a token describing the sort key, e.g., ctime, atime, size.
KEY_CMP_FUNC is a function to compare records based on that key, e.g.,
ctime_cmp, atime_cmp, size_cmp. Append KEY_NAME to the string,
'[rev_]xstr{cmp|coll}[_df]_', to create each function name. */
#define DEFINE_SORT_FUNCTIONS(key_name, key_cmp_func) \
/* direct, non-dirfirst versions */ \
static int xstrcoll_##key_name (V a, V b) \
{ return key_cmp_func (a, b, xstrcoll); } \
ATTRIBUTE_PURE static int strcmp_##key_name (V a, V b) \
{ return key_cmp_func (a, b, strcmp); } \
 \
/* reverse, non-dirfirst versions */ \
static int rev_xstrcoll_##key_name (V a, V b) \
{ return key_cmp_func (b, a, xstrcoll); } \
ATTRIBUTE_PURE static int rev_strcmp_##key_name (V a, V b) \
{ return key_cmp_func (b, a, strcmp); } \
 \
/* direct, dirfirst versions */ \
static int xstrcoll_df_##key_name (V a, V b) \
{ return dirfirst_check (a, b, xstrcoll_##key_name); } \
ATTRIBUTE_PURE static int strcmp_df_##key_name (V a, V b) \
{ return dirfirst_check (a, b, strcmp_##key_name); } \
 \
/* reverse, dirfirst versions */ \
static int rev_xstrcoll_df_##key_name (V a, V b) \
{ return dirfirst_check (a, b, rev_xstrcoll_##key_name); } \
ATTRIBUTE_PURE static int rev_strcmp_df_##key_name (V a, V b) \
{ return dirfirst_check (a, b, rev_strcmp_##key_name); }

static int
cmp_ctime (struct fileinfo const *a, struct fileinfo const *b,
           int (*cmp) (char const *, char const *))
{
int diff = timespec_cmp (get_stat_ctime (&b->stat),
                        get_stat_ctime (&a->stat));
return diff ? diff : cmp (a->name, b->name);
}

static int
cmp_mtime (struct fileinfo const *a, struct fileinfo const *b,
           int (*cmp) (char const *, char const *))
{
int diff = timespec_cmp (get_stat_mtime (&b->stat),
                        get_stat_mtime (&a->stat));
return diff ? diff : cmp (a->name, b->name);
}

static int
cmp_atime (struct fileinfo const *a, struct fileinfo const *b,
           int (*cmp) (char const *, char const *))

```

```

{
int diff = timespec_cmp (get_stat_atime (&b->stat),
                        get_stat_atime (&a->stat));
return diff ? diff : cmp (a->name, b->name);
}

static int
cmp_btime (struct fileinfo const *a, struct fileinfo const *b,
           int (*cmp) (char const *, char const *))
{
int diff = timespec_cmp (get_stat_btime (&b->stat),
                        get_stat_btime (&a->stat));
return diff ? diff : cmp (a->name, b->name);
}

static int
off_cmp (off_t a, off_t b)
{
return (a > b) - (a < b);
}

static int
cmp_size (struct fileinfo const *a, struct fileinfo const *b,
          int (*cmp) (char const *, char const *))
{
int diff = off_cmp (b->stat.st_size, a->stat.st_size);
return diff ? diff : cmp (a->name, b->name);
}

static int
cmp_name (struct fileinfo const *a, struct fileinfo const *b,
          int (*cmp) (char const *, char const *))
{
return cmp (a->name, b->name);
}

/* Compare file extensions. Files with no extension are 'smallest'.
If extensions are the same, compare by file names instead. */

static int
cmp_extension (struct fileinfo const *a, struct fileinfo const *b,
               int (*cmp) (char const *, char const *))
{
char const *base1 = strrchr (a->name, '.');
char const *base2 = strrchr (b->name, '.');
int diff = cmp (base1 ? base1 : "", base2 ? base2 : "");
return diff ? diff : cmp (a->name, b->name);
}

/* Return the (cached) screen width,
for the NAME associated with the passed fileinfo F. */

static size_t
fileinfo_name_width (struct fileinfo const *f)
{
return f->width
    ? f->width
    : quote_name_width (f->name, filename_quoting_options, f->quoted);
}

```

```

static int
cmp_width (struct fileinfo const *a, struct fileinfo const *b,
           int (*cmp) (char const *, char const *))
{
    int diff = fileinfo_name_width (a) - fileinfo_name_width (b);
    return diff ? diff : cmp (a->name, b->name);
}

DEFINE_SORT_FUNCTIONS (ctime, cmp_ctime)
DEFINE_SORT_FUNCTIONS (mtime, cmp_mtime)
DEFINE_SORT_FUNCTIONS (atime, cmp_atime)
DEFINE_SORT_FUNCTIONS (btime, cmp_btime)
DEFINE_SORT_FUNCTIONS (size, cmp_size)
DEFINE_SORT_FUNCTIONS (name, cmp_name)
DEFINE_SORT_FUNCTIONS (extension, cmp_extension)
DEFINE_SORT_FUNCTIONS (width, cmp_width)

```

*/\* Compare file versions.*

Unlike the other compare functions, cmp\_version does not fail because filevercmp and strcmp do not fail; cmp\_version uses strcmp instead of xstrcoll because filevercmp is locale-independent so strcmp is its appropriate secondary.

All the other sort options need xstrcoll and strcmp variants, because they all use xstrcoll (either as the primary or secondary sort key), and xstrcoll has the ability to do a longjmp if strcoll fails for locale reasons. \*/

```

static int
cmp_version (struct fileinfo const *a, struct fileinfo const *b)
{
    int diff = filevercmp (a->name, b->name);
    return diff ? diff : strcmp (a->name, b->name);
}

static int
xstrcoll_version (V a, V b)
{
    return cmp_version (a, b);
}
static int
rev_xstrcoll_version (V a, V b)
{
    return cmp_version (b, a);
}
static int
xstrcoll_df_version (V a, V b)
{
    return dirfirst_check (a, b, xstrcoll_version);
}
static int
rev_xstrcoll_df_version (V a, V b)
{
    return dirfirst_check (a, b, rev_xstrcoll_version);
}
```

*/\* We have 2^3 different variants for each sort-key function (for 3 independent sort modes).*

The function pointers stored in this array must be dereferenced as:

```
sort_variants[sort_key][use_strcmp][reverse][dirs_first]
```

Note that the order in which sort keys are listed in the function pointer array below is defined by the order of the elements in the time\_type and sort\_type enums! \*/

```
#define LIST_SORTFUNCTION_VARIANTS(key_name) \
{ \
    { xstrcoll_##key_name, xstrcoll_df_##key_name }, \
    { rev_xstrcoll_##key_name, rev_xstrcoll_df_##key_name }, \
}, \
{ \
    { strcmp_##key_name, strcmp_df_##key_name }, \
    { rev_strcmp_##key_name, rev_strcmp_df_##key_name }, \
} \
}

static qsortFunc const sort_functions[][2][2][2] = \
{
    LIST_SORTFUNCTION_VARIANTS (name),
    LIST_SORTFUNCTION_VARIANTS (extension),
    LIST_SORTFUNCTION_VARIANTS (width),
    LIST_SORTFUNCTION_VARIANTS (size),

    {
        {
            { xstrcoll_version, xstrcoll_df_version },
            { rev_xstrcoll_version, rev_xstrcoll_df_version },
        },
        /* We use nullptr for the strcmp variants of version comparison
           since as explained in cmp_version definition, version comparison
           does not rely on xstrcoll, so it will never longjmp, and never
           need to try the strcmp fallback. */
        {
            { nullptr, nullptr },
            { nullptr, nullptr },
        }
    },
    /* last are time sort functions */
    LIST_SORTFUNCTION_VARIANTS (mtime),
    LIST_SORTFUNCTION_VARIANTS (ctime),
    LIST_SORTFUNCTION_VARIANTS (atime),
    LIST_SORTFUNCTION_VARIANTS (btime)
};

/* The number of sort keys is calculated as the sum of
   the number of elements in the sort_type enum (i.e., sort_numtypes)
   -2 because neither sort_time nor sort_none use entries themselves
   the number of elements in the time_type enum (i.e., time_numtypes)
This is because when sort_type==sort_time, we have up to
time_numtypes possible sort keys.
```

This line verifies at compile-time that the array of sort functions has been initialized for all possible sort keys. \*/

```
static_assert (ARRAY_CARDINALITY (sort_functions)
              == sort_numtypes - 2 + time_numtypes);
```

```

/* Set up SORTED_FILE to point to the in-use entries in CWD_FILE, in order. */

static void
initialize_ordering_vector (void)
{
for (size_t i = 0; i < cwd_n_used; i++)
    sorted_file[i] = &cwd_file[i];
}

/* Cache values based on attributes global to all files. */

static void
update_current_files_info (void)
{
/* Cache screen width of name, if needed multiple times. */
if (sort_type == sort_width
    || (line_length && (format == many_per_line || format == horizontal)))
{
size_t i;
for (i = 0; i < cwd_n_used; i++)
{
    struct fileinfo *f = sorted_file[i];
    f->width = fileinfo_name_width (f);
}
}
}

/* Sort the files now in the table. */

static void
sort_files (void)
{
bool use_strcmp;

if (sorted_file_alloc < cwd_n_used + cwd_n_used / 2)
{
    free (sorted_file);
    sorted_file = xnmalloc (cwd_n_used, 3 * sizeof *sorted_file);
    sorted_file_alloc = 3 * cwd_n_used;
}

initialize_ordering_vector ();

update_current_files_info ();

if (sort_type == sort_none)
    return;

/* Try strcoll. If it fails, fall back on strcmp. We can't safely
 ignore strcoll failures, as a failing strcoll might be a
 comparison function that is not a total order, and if we ignored
 the failure this might cause qsort to dump core. */

if (! setjmp (failed_strcoll))
    use_strcmp = false; /* strcoll() succeeded */
else
{
    use_strcmp = true;
    affirm (sort_type != sort_version);
    initialize_ordering_vector ();
}

```

```

    }

/* When sort_type == sort_time, use time_type as subindex. */
mpsort ((void const **) sorted_file, cwd_n_used,
    sort_functions[sort_type + (sort_type == sort_time ? time_type : 0)]
        [use_strcmp][sort_reverse]
        [directories_first]);
}

/* List all the files now in the table. */

static void
print_current_files (void)
{
size_t i;

switch (format)
{
case one_per_line:
for (i = 0; i < cwd_n_used; i++)
{
    print_file_name_and_frills (sorted_file[i], 0);
    putchar (eolbyte);
}
break;

case many_per_line:
if (!line_length)
    print_with_separator (' ');
else
    print_many_per_line ();
break;

case horizontal:
if (!line_length)
    print_with_separator (' ');
else
    print_horizontal ();
break;

case with_commas:
print_with_separator (',');
break;

case long_format:
for (i = 0; i < cwd_n_used; i++)
{
    set_normal_color ();
    print_long_format (sorted_file[i]);
    dired_outbyte (eolbyte);
}
break;
}

/* Replace the first %b with precomputed aligned month names.
Note on glibc-2.7 at least, this speeds up the whole 'ls -lU'
process by around 17%, compared to letting strftime() handle the %b. */

static size_t
```

```

align_nstrftime (char *buf, size_t size, bool recent, struct tm const *tm,
                timezone_t tz, int ns)
{
    char const *nfmt = (use_abformat
                        ? abformat[recent][tm->tm_mon]
                        : long_time_format[recent]);
    return strftime (buf, size, nfmt, tm, tz, ns);
}

/* Return the expected number of columns in a long-format timestamp,
or zero if it cannot be calculated. */

static int
long_time_expected_width (void)
{
    static int width = -1;

    if (width < 0)
    {
        time_t epoch = 0;
        struct tm tm;
        char buf[TIME_STAMP_LEN_MAXIMUM + 1];

        /* In case you're wondering if localtime_rz can fail with an input time_t
         value of 0, let's just say it's very unlikely, but not inconceivable.
         The TZ environment variable would have to specify a time zone that
         is 2**31-1900 years or more ahead of UTC. This could happen only on
         a 64-bit system that blindly accepts e.g., TZ=UTC+2000000000000000.
         However, this is not possible with Solaris 10 or glibc-2.3.5, since
         their implementations limit the offset to 167:59 and 24:00, resp. */
        if (localtime_rz (localtz, &epoch, &tm))
        {
            size_t len = align_nstrftime (buf, sizeof buf, false,
                                         &tm, localtz, 0);
            if (len != 0)
                width = mbsnwidth (buf, len, MBSWIDTH_FLAGS);
        }
        if (width < 0)
            width = 0;
    }

    return width;
}

/* Print the user or group name NAME, with numeric id ID, using a
print width of WIDTH columns. */

static void
format_user_or_group (char const *name, uintmax_t id, int width)
{
    if (name)
    {
        int name_width = mbswidth (name, MBSWIDTH_FLAGS);
        int width_gap = name_width < 0 ? 0 : width - name_width;
        int pad = MAX (0, width_gap);
        dired_outstring (name);

        do
            dired_outbyte (' ');
    }
}

```

```

        while (pad--);
    }
else
    dired_pos += printf ("%*ju ", width, id);
}

/* Print the name or id of the user with id U, using a print width of
WIDTH. */

static void
format_user (uid_t u, int width, bool stat_ok)
{
format_user_or_group (! stat_ok ? "?" :
(numeric_ids ? nullptr : getuser (u)), u, width);
}

/* Likewise, for groups. */

static void
format_group (gid_t g, int width, bool stat_ok)
{
format_user_or_group (! stat_ok ? "?" :
(numeric_ids ? nullptr : getgroup (g)), g, width);
}

/* Return the number of columns that format_user_or_group will print,
or -1 if unknown. */

static int
format_user_or_group_width (char const *name, uintmax_t id)
{
return (name
? mbswidth (name, MBSWIDTH_FLAGS)
: snprintf (nullptr, 0, "%ju", id));
}

/* Return the number of columns that format_user will print,
or -1 if unknown. */

static int
format_user_width (uid_t u)
{
return format_user_or_group_width (numeric_ids ? nullptr : getuser (u), u);
}

/* Likewise, for groups. */

static int
format_group_width (gid_t g)
{
return format_user_or_group_width (numeric_ids ? nullptr : getgroup (g), g);
}

/* Return a pointer to a formatted version of F->stat.st_ino,
possibly using buffer, which must be at least
INT_BUFSIZE_BOUND (uintmax_t) bytes. */
static char *
format_inode (char buf[INT_BUFSIZE_BOUND (uintmax_t)],
const struct fileinfo *f)
{

```

```

return (f->stat_ok && f->stat.st_ino != NOT_AN_INODE_NUMBER
    ? umaxtostr (f->stat.st_ino, buf)
    : (char *) "?");
}

/* Print information about F in long format. */
static void
print_long_format (const struct fileinfo *f)
{
char modebuf[12];
char buf
    [LONGEST_HUMAN_READABLE + 1          /* inode */
     + LONGEST_HUMAN_READABLE + 1        /* size in blocks */
     + sizeof (modebuf) - 1 + 1         /* mode string */
     + INT_BUFSIZE_BOUND (uintmax_t)    /* st_nlink */
     + LONGEST_HUMAN_READABLE + 2       /* major device number */
     + LONGEST_HUMAN_READABLE + 1       /* minor device number */
     + TIME_STAMP_LEN_MAXIMUM + 1      /* max length of time/date */
];
size_t s;
char *p;
struct timespec when_timespec;
struct tm when_local;
bool btime_ok = true;

/* Compute the mode string, except remove the trailing space if no
   file in this directory has an ACL or security context. */
if (f->stat_ok)
    filemodestring (&f->stat, modebuf);
else
{
    modebuf[0] = filetype_letter[f->filetype];
    memset (modebuf + 1, '?', 10);
    modebuf[11] = '\0';
}
if (! any_has_acl)
    modebuf[10] = '\0';
else if (f->acl_type == ACL_T_LSM_CONTEXT_ONLY)
    modebuf[10] = '.';
else if (f->acl_type == ACL_T_YES)
    modebuf[10] = '+';

switch (time_type)
{
case time_ctime:
when_timespec = get_stat_ctime (&f->stat);
break;
case time_mtime:
when_timespec = get_stat_mtime (&f->stat);
break;
case time_atime:
when_timespec = get_stat_atime (&f->stat);
break;
case time_btime:
when_timespec = get_stat_btime (&f->stat);
if (when_timespec.tv_sec == -1 && when_timespec.tv_nsec == -1)
    btime_ok = false;
break;
default:
unreachable ();
}

```

```

    }

p = buf;

if (print_inode)
{
    char hbuf[INT_BUFSIZE_BOUND (uintmax_t)];
    p += sprintf (p, "%*s ", inode_number_width, format_inode (hbuf, f));
}

if (print_block_size)
{
    char hbuf[LONGEST_HUMAN_READABLE + 1];
    char const *blocks =
        (! f->stat_ok
         ? "?"
           : human_readable (STP_NBLOCKS (&f->stat), hbuf, human_output_opts,
                             ST_NBLOCKSIZE, output_block_size));
    int blocks_width = mbswidth (blocks, MBSWIDTH_FLAGS);
    for (int pad = blocks_width < 0 ? 0 : block_size_width - blocks_width;
         0 < pad; pad--)
        *p++ = ' ';
    while ((*p++ = *blocks++) != '\0')
        continue;
    p[-1] = ' ';
}

/* The last byte of the mode string is the POSIX
   "optional alternate access method flag". */
{
    char hbuf[INT_BUFSIZE_BOUND (uintmax_t)];
    p += sprintf (p, "%s %*s ", modebuf, nlink_width,
                  ! f->stat_ok ? "?" : umaxtostr (f->stat.st_nlink, hbuf));
}

dired_indent ();

if (print_owner || print_group || print_author || print_scontext)
{
    dired_outbuf (buf, p - buf);

    if (print_owner)
        format_user (f->stat.st_uid, owner_width, f->stat_ok);

    if (print_group)
        format_group (f->stat.st_gid, group_width, f->stat_ok);

    if (print_author)
        format_user (f->stat.st_author, author_width, f->stat_ok);

    if (print_scontext)
        format_user_or_group (f->scontext, 0, scontext_width);
}

p = buf;
}

if (f->stat_ok
    && (S_ISCHR (f->stat.st_mode) || S_ISBLK (f->stat.st_mode)))
{
    char majorbuf[INT_BUFSIZE_BOUND (uintmax_t)];

```

```

char minorbuf[INT_BUFSIZE_BOUND (uintmax_t)];
int blanks_width = (file_size_width
    - (major_device_number_width + 2
        + minor_device_number_width));
p += sprintf (p, "%*s, %*s ",
    major_device_number_width + MAX (0, blanks_width),
    umaxtostr (major (f->stat.st_rdev), majorbuf),
    minor_device_number_width,
    umaxtostr (minor (f->stat.st_rdev), minorbuf));
}
else
{
    char hbuf[LONGEST_HUMAN_READABLE + 1];
    char const *size =
        (! f->stat_ok
        ? "?" :
            : human_readable (unsigned_file_size (f->stat.st_size),
                hbuf, file_human_output_opts, 1,
                file_output_block_size));
    int size_width = mbswidth (size, MBSWIDTH_FLAGS);
    for (int pad = size_width < 0 ? 0 : file_size_width - size_width;
        0 < pad; pad--)
        *p++ = ' ';
    while ((*p++ = *size++))
        continue;
    p[-1] = ' ';
}

s = 0;
*p = '\1';

if (f->stat_ok && btime_ok
    && localtime_rz (localtz, &when_timespec.tv_sec, &when_local))
{
    struct timespec six_months_ago;
    bool recent;

    /* If the file appears to be in the future, update the current
       time, in case the file happens to have been modified since
       the last time we checked the clock. */
    if (timespec_cmp (current_time, when_timespec) < 0)
        gettimeofday (&current_time);

    /* Consider a time to be recent if it is within the past six months.
       A Gregorian year has 365.2425 * 24 * 60 * 60 == 31556952 seconds
       on the average. Write this value as an integer constant to
       avoid floating point hassles. */
    six_months_ago.tv_sec = current_time.tv_sec - 31556952 / 2;
    six_months_ago.tv_nsec = current_time.tv_nsec;

    recent = (timespec_cmp (six_months_ago, when_timespec) < 0
        && timespec_cmp (when_timespec, current_time) < 0);

    /* We assume here that all time zones are offset from UTC by a
       whole number of seconds. */
    s = align_nstrftime (p, TIME_STAMP_LEN_MAXIMUM + 1, recent,
        &when_local, localtz, when_timespec.tv_nsec);
}

if (s || !*p)

```

```

    {
        p += s;
        *p++ = ' ';
    }
else
{
    /* The time cannot be converted using the desired format, so
       print it as a huge integer number of seconds. */
    char hbuf[INT_BUFSIZE_BOUND (intmax_t)];
    p += sprintf (p, "%*s", long_time_expected_width (),
                  (! f->stat_ok || ! btime_ok
                   ? "?" :
                   : timetostr (when_timespec.tv_sec, hbuf)));
    /* FIXME: (maybe) We discarded when_timespec.tv_nsec. */
}
}

dired_outbuf (buf, p - buf);
size_t w = print_name_with_quoting (f, false, &dired_obstack, p - buf);

if (f->filetype == symbolic_link)
{
    if (f->linkname)
    {
        dired_outstring (" -> ");
        print_name_with_quoting (f, true, nullptr, (p - buf) + w + 4);
        if (indicator_style != none)
            print_type_indicator (true, f->linkmode, unknown);
    }
}
else if (indicator_style != none)
    print_type_indicator (f->stat_ok, f->stat.st_mode, f->filetype);
}

/* Write to *BUF a quoted representation of the file name NAME, if non-null,
   using OPTIONS to control quoting. *BUF is set to NAME if no quoting
   is required. *BUF is allocated if more space required (and the original
   *BUF is not deallocated).
   Store the number of screen columns occupied by NAME's quoted
   representation into WIDTH, if non-null.
   Store into PAD whether an initial space is needed for padding.
   Return the number of bytes in *BUF. */

static size_t
quote_name_buf (char **inbuf, size_t bufsize, char *name,
                struct quoting_options const *options,
                int needs_general_quoting, size_t *width, bool *pad)
{
    char *buf = *inbuf;
    size_t displayed_width IF_LINT ( = 0);
    size_t len = 0;
    bool quoted;

    enum quoting_style qs = get_quoting_style (options);
    bool needs_further_quoting = qmark_funny_chars
        && (qs == shell_quoting_style
              || qs == shell_always_quoting_style
              || qs == literal_quoting_style);

    if (needs_general_quoting != 0)
    {

```

```

len = quotearg_buffer (buf, bufsize, name, -1, options);
if (bufsize <= len)
{
    buf = xmalloc (len + 1);
    quotearg_buffer (buf, len + 1, name, -1, options);
}

quoted = (*name != *buf) || strlen (name) != len;
}
else if (needs_further_quoting)
{
    len = strlen (name);
    if (bufsize <= len)
        buf = xmalloc (len + 1);
    memcpy (buf, name, len + 1);

    quoted = false;
}
else
{
    len = strlen (name);
    buf = name;
    quoted = false;
}

if (needs_further_quoting)
{
    if (MB_CUR_MAX > 1)
    {
        char const *p = buf;
        char const *plimit = buf + len;
        char *q = buf;
        displayed_width = 0;

        while (p < plimit)
            switch (*p)
            {
                case ' ': case '!': case "'": case '#': case '%':
                case '&': case '\\': case '(': case ')': case '*':
                case '+': case ',': case '-': case '.': case '/':
                case '0': case '1': case '2': case '3': case '4':
                case '5': case '6': case '7': case '8': case '9':
                case ',': case ',': case '<': case '=': case '>':
                case '?':
                case 'A': case 'B': case 'C': case 'D': case 'E':
                case 'F': case 'G': case 'H': case 'I': case 'J':
                case 'K': case 'L': case 'M': case 'N': case 'O':
                case 'P': case 'Q': case 'R': case 'S': case 'T':
                case 'U': case 'V': case 'W': case 'X': case 'Y':
                case 'Z':
                case '[': case '\\': case ']': case '^': case '_':
                case 'a': case 'b': case 'c': case 'd': case 'e':
                case 'f': case 'g': case 'h': case 'i': case 'j':
                case 'k': case 'l': case 'm': case 'n': case 'o':
                case 'p': case 'q': case 'r': case 's': case 't':
                case 'u': case 'v': case 'w': case 'x': case 'y':
                case 'z': case '{': case '|': case '}': case '~':
/* These characters are printable ASCII characters. */
                *q++ = *p++;
                displayed_width += 1;
            }
        }
    }
}

```

```

break;
default:
/* If we have a multibyte sequence, copy it until we
   reach its end, replacing each non-printable multibyte
   character with a single question mark. */
{
    mbstate_t mbstate; mbszero (&mbstate);
    do
    {
        char32_t wc;
        size_t bytes;
        int w;

        bytes = mbrtoc32 (&wc, p, plimit - p, &mbstate);

        if (bytes == (size_t) -1)
        {
            /* An invalid multibyte sequence was
               encountered. Skip one input byte, and
               put a question mark. */
            p++;
            *q++ = '?';
            displayed_width += 1;
            break;
        }

        if (bytes == (size_t) -2)
        {
            /* An incomplete multibyte character
               at the end. Replace it entirely with
               a question mark. */
            p = plimit;
            *q++ = '?';
            displayed_width += 1;
            break;
        }

        if (bytes == 0)
        /* A null wide character was encountered. */
        bytes = 1;

        w = c32width (wc);
        if (w >= 0)
        {
            /* A printable multibyte character.
               Keep it. */
            for (; bytes > 0; --bytes)
                *q++ = *p++;
            displayed_width += w;
        }
        else
        {
            /* An nonprintable multibyte character.
               Replace it entirely with a question
               mark. */
            p += bytes;
            *q++ = '?';
            displayed_width += 1;
        }
    }
}

```

```

        while (! mbsinit (&mbstate));
    }
    break;
}

/* The buffer may have shrunk. */
len = q - buf;
}
else
{
    char *p = buf;
    char const *plimit = buf + len;

    while (p < plimit)
    {
        if (! isprint (to_uchar (*p)))
            *p = '?';
        p++;
    }
    displayed_width = len;
}
else if (width != nullptr)
{
    if (MB_CUR_MAX > 1)
    {
        displayed_width = mbsnwidth (buf, len, MBSWIDTH_FLAGS);
        displayed_width = MAX (0, displayed_width);
    }
    else
    {
        char const *p = buf;
        char const *plimit = buf + len;

        displayed_width = 0;
        while (p < plimit)
        {
            if (isprint (to_uchar (*p)))
                displayed_width++;
            p++;
        }
    }
}

/* Set padding to better align quoted items,
   and also give a visual indication that quotes are
   not actually part of the name. */
*pad = (align_variable_outer_quotes && cwd_some_quoted && ! quoted);

if (width != nullptr)
    *width = displayed_width;

*inbuf = buf;

return len;
}

static size_t
quote_name_width (char const *name, struct quoting_options const *options,
                  int needs_general_quoting)

```

```

{
char smallbuf[BUFSIZ];
char *buf = smallbuf;
size_t width;
bool pad;

quote_name_buf (&buf, sizeof smallbuf, (char *) name, options,
    needs_general_quoting, &width, &pad);

if (buf != smallbuf && buf != name)
    free (buf);

width += pad;

return width;
}

/* %XX escape any input out of range as defined in RFC3986,
and also if PATH, convert all path separators to '/'. */
static char *
file_escape (char const *str, bool path)
{
char *esc = xmalloc (3, strlen (str) + 1);
char *p = esc;
while (*str)
{
    if (path && ISSLASH (*str))
    {
        *p++ = '/';
        str++;
    }
    else if (RFC3986[to_uchar (*str)])
        *p++ = *str++;
    else
        p += sprintf (p, "%%%%02x", to_uchar (*str++));
}
*p = '\0';
return esc;
}

static size_t
quote_name (char const *name, struct quoting_options const *options,
    int needs_general_quoting, const struct bin_str *color,
    bool allow_pad, struct obstack *stack, char const *absolute_name)
{
char smallbuf[BUFSIZ];
char *buf = smallbuf;
size_t len;
bool pad;

len = quote_name_buf (&buf, sizeof smallbuf, (char *) name, options,
    needs_general_quoting, nullptr, &pad);

if (pad && allow_pad)
    dired_outbyte (' ');

if (color)
    print_color_indicator (color);

/* If we're padding, then don't include the outer quotes in

```

```

    the --hyperlink, to improve the alignment of those links. */
bool skip_quotes = false;

if (absolute_name)
{
    if (align_variable_outer_quotes && cwd_some_quoted && ! pad)
    {
        skip_quotes = true;
        putchar (*buf);
    }
    char *h = file_escape (hostname, /* path= */ false);
    char *n = file_escape (absolute_name, /* path= */ true);
    /* TODO: It would be good to be able to define parameters
       to give hints to the terminal as how best to render the URI.
       For example since ls is outputting a dense block of URIs
       it would be best to not underline by default, and only
       do so upon hover etc. */
    printf ("\033]8;;file://%"S "%s\033[0m", h, *n == '/' ? "" : "/", n);
    free (h);
    free (n);
}

if (stack)
    push_current_dired_pos (stack);

fwrite (buf + skip_quotes, 1, len - (skip_quotes * 2), stdout);

dired_pos += len;

if (stack)
    push_current_dired_pos (stack);

if (absolute_name)
{
    fputs ("\033]8;\033[0m", stdout);
    if (skip_quotes)
        putchar (*(buf + len - 1));
}

if (buf != smallbuf && buf != name)
    free (buf);

return len + pad;
}

static size_t
print_name_with_quoting (const struct fileinfo *f,
                        bool symlink_target,
                        struct obstack *stack,
                        size_t start_col)
{
char const *name = symlink_target ? f->linkname : f->name;

const struct bin_str *color
= print_with_color ? get_color_indicator (f, symlink_target) : nullptr;

bool used_color_this_time = (print_with_color
&& (color || is_colored (C_NORM)));

size_t len = quote_name (name, filename_quoting_options, f->quoted,

```

```

        color, !symlink_target, stack, f->absolute_name);

process_signals ();
if (used_color_this_time)
{
    prep_non_filename_text ();

/* We use the byte length rather than display width here as
   an optimization to avoid accurately calculating the width,
   because we only output the clear to EOL sequence if the name
   _might_ wrap to the next line. This may output a sequence
   unnecessarily in multi-byte locales for example,
   but in that case it's inconsequential to the output. */
if (line_length
    && (start_col / line_length != (start_col + len - 1) / line_length))
    put_indicator (&color_indicator[C_CLR_TO_EOL]);
}

return len;
}

static void
prep_non_filename_text (void)
{
if (color_indicator[C_END].string != nullptr)
    put_indicator (&color_indicator[C_END]);
else
{
    put_indicator (&color_indicator[C_LEFT]);
    put_indicator (&color_indicator[C_RESET]);
    put_indicator (&color_indicator[C_RIGHT]);
}
}

/* Print the file name of 'f' with appropriate quoting.
Also print file size, inode number, and filetype indicator character,
as requested by switches. */

static size_t
print_file_name_and_frills (const struct fileinfo *f, size_t start_col)
{
char buf[MAX (LONGEST_HUMAN_READABLE + 1, INT_BUFSIZE_BOUND (uintmax_t))];

set_normal_color ();

if (print_inode)
    printf ("%*s ", format == with_commas ? 0 : inode_number_width,
            format_inode (buf, f));

if (print_block_size)
    printf ("%*s ", format == with_commas ? 0 : block_size_width,
            ! f->stat_ok ? "?" :
            : human_readable (STP_NBLOCKS (&f->stat), buf, human_output_opts,
                             ST_NBLOCKSIZE, output_block_size));

if (print_scontext)
    printf ("%*s ", format == with_commas ? 0 : scontext_width, f->scontext);

size_t width = print_name_with_quoting (f, false, nullptr, start_col);

```

```

if (indicator_style != none)
    width += print_type_indicator (f->stat_ok, f->stat.st_mode, f->filetype);

return width;
}

/* Given these arguments describing a file, return the single-byte
type indicator, or 0. */
static char
get_type_indicator (bool stat_ok, mode_t mode, enum filetype type)
{
char c;

if (stat_ok ? S_ISREG (mode) : type == normal)
{
    if (stat_ok && indicator_style == classify && (mode & S_IXUGO))
        c = '*';
    else
        c = 0;
}
else
{
    if (stat_ok ? S_ISDIR (mode) : type == directory || type == arg_directory)
        c = '/';
    else if (indicator_style == slash)
        c = 0;
    else if (stat_ok ? S_ISLNK (mode) : type == symbolic_link)
        c = '@';
    else if (stat_ok ? S_ISFIFO (mode) : type == fifo)
        c = '|';
    else if (stat_ok ? S_ISSOCK (mode) : type == sock)
        c = '=';
    else if (stat_ok && S_ISDOOR (mode))
        c = '>';
    else
        c = 0;
}
return c;
}

static bool
print_type_indicator (bool stat_ok, mode_t mode, enum filetype type)
{
char c = get_type_indicator (stat_ok, mode, type);
if (c)
    dired_outbyte (c);
return !c;
}

/* Returns if color sequence was printed. */
static bool
print_color_indicator (const struct bin_str *ind)
{
if (ind)
{
    /* Need to reset so not dealing with attribute combinations */
    if (is_colored (C_NORM))
        restore_default_color ();
    put_indicator (&color_indicator[C_LEFT]);
    put_indicator (ind);
}

```

```

        put_indicator (&color_indicator[C_RIGHT]);
    }

return ind != nullptr;
}

/* Returns color indicator or nullptr if none. */
ATTRIBUTE_PURE
static const struct bin_str*
get_color_indicator (const struct fileinfo *f, bool symlink_target)
{
enum indicator_no type;
struct color_ext_type *ext; /* Color extension */
size_t len; /* Length of name */

char const *name;
mode_t mode;
int linkok;
if (symlink_target)
{
    name = f->linkname;
    mode = f->linkmode;
    linkok = f->linkok ? 0 : -1;
}
else
{
    name = f->name;
    mode = file_or_link_mode (f);
    linkok = f->linkok;
}

/* Is this a nonexistent file? If so, linkok == -1. */

if (linkok == -1 && is_colored (C_MISSING))
    type = C_MISSING;
else if (!f->stat_ok)
{
    static enum indicator_no filetype_indicator[] = FILETYPE_INDICATORS;
    type = filetype_indicator[f->filetype];
}
else
{
    if (S_ISREG (mode))
    {
        type = C_FILE;

        if ((mode & S_ISUID) != 0 && is_colored (C_SETUID))
            type = C_SETUID;
        else if ((mode & S_ISGID) != 0 && is_colored (C_SETGID))
            type = C_SETGID;
        else if (is_colored (C_CAP) && f->has_capability)
            type = C_CAP;
        else if ((mode & S_IXUGO) != 0 && is_colored (C_EXEC))
            type = C_EXEC;
        else if ((1 < f->stat.st_nlink) && is_colored (C_MULTIHARDLINK))
            type = C_MULTIHARDLINK;
    }
    else if (S_ISDIR (mode))
    {
        type = C_DIR;
    }
}

```

```

    if ((mode & S_ISVTX) && (mode & S_IWOTH)
        && is_colored (C_STICKY_OTHER_WRITABLE))
        type = C_STICKY_OTHER_WRITABLE;
    else if ((mode & S_IWOTH) != 0 && is_colored (C_OTHER_WRITABLE))
        type = C_OTHER_WRITABLE;
    else if ((mode & S_ISVTX) != 0 && is_colored (C_STICKY))
        type = C_STICKY;
    }
else if (S_ISLNK (mode))
    type = C_LINK;
else if (S_ISFIFO (mode))
    type = C_FIFO;
else if (S_ISSOCK (mode))
    type = C_SOCK;
else if (S_ISBLK (mode))
    type = C_BLK;
else if (S_ISCHR (mode))
    type = C_CHR;
else if (S_ISDOOR (mode))
    type = C_DOOR;
else
{
    /* Classify a file of some other type as C_ORPHAN. */
    type = C_ORPHAN;
}
}

/* Check the file's suffix only if still classified as C_FILE. */
ext = nullptr;
if (type == C_FILE)
{
    /* Test if NAME has a recognized suffix. */

    len = strlen (name);
    name += len;           /* Pointer to final \0. */
    for (ext = color_ext_list; ext != nullptr; ext = ext->next)
    {
        if (ext->ext.len <= len)
        {
            if (ext->exact_match)
            {
                if (STREQ_LEN (name - ext->ext.len, ext->ext.string,
                               ext->ext.len))
                    break;
            }
            else
            {
                if (c_strncasecmp (name - ext->ext.len, ext->ext.string,
                                   ext->ext.len) == 0)
                    break;
            }
        }
    }
}

/* Adjust the color for orphaned symlinks. */
if (type == C_LINK && !linkok)
{
    if (color_symlink_as_referent || is_colored (C_ORPHAN))

```

```

        type = C_ORPHAN;
    }

const struct bin_str *const s
= ext ? &(ext->seq) : &color_indicator[type];

return s->string ? s : nullptr;
}

/* Output a color indicator (which may contain nulls). */
static void
put_indicator (const struct bin_str *ind)
{
if (! used_color)
{
    used_color = true;

/* If the standard output is a controlling terminal, watch out
   for signals, so that the colors can be restored to the
   default state if "ls" is suspended or interrupted. */

if (0 <= tcgetpgrp (STDOUT_FILENO))
    signal_init ();

prep_non_filename_text ();
}

fwrite (ind->string, ind->len, 1, stdout);
}

static size_t
length_of_file_name_and_frills (const struct fileinfo *f)
{
size_t len = 0;
char buf[MAX (LONGEST_HUMAN_READABLE + 1, INT_BUFSIZE_BOUND (uintmax_t))];

if (print_inode)
    len += 1 + (format == with_commas
                 ? strlen (umaxtostr (f->stat.st_ino, buf))
                 : inode_number_width);

if (print_block_size)
    len += 1 + (format == with_commas
                 ? strlen (! f->stat_ok ? "?" :
                           : human_readable (STP_NBLOCKS (&f->stat), buf,
                                             human_output_opts, ST_NBLOCKSIZE,
                                             output_block_size))
                 : block_size_width);

if (print_scontext)
    len += 1 + (format == with_commas ? strlen (f->scontext) : scontext_width);

len += fileinfo_name_width (f);

if (indicator_style != none)
{
    char c = get_type_indicator (f->stat_ok, f->stat.st_mode, f->filetype);
    len += (c != 0);
}

```

```

return len;
}

static void
print_many_per_line (void)
{
size_t row;           /* Current row. */
size_t cols = calculate_columns (true);
struct column_info const *line_fmt = &column_info[cols - 1];

/* Calculate the number of rows that will be in each column except possibly
   for a short column on the right. */
size_t rows = cwd_n_used / cols + (cwd_n_used % cols != 0);

for (row = 0; row < rows; row++)
{
    size_t col = 0;
    size_t filesno = row;
    size_t pos = 0;

    /* Print the next row. */
    while (true)
    {
        struct fileinfo const *f = sorted_file[filesno];
        size_t name_length = length_of_file_name_and_frills (f);
        size_t max_name_length = line_fmt->col_arr[col++];
        print_file_name_and_frills (f, pos);

        filesno += rows;
        if (filesno >= cwd_n_used)
            break;

        indent (pos + name_length, pos + max_name_length);
        pos += max_name_length;
    }
    putchar (eolbyte);
}
}

static void
print_horizontal (void)
{
size_t filesno;
size_t pos = 0;
size_t cols = calculate_columns (false);
struct column_info const *line_fmt = &column_info[cols - 1];
struct fileinfo const *f = sorted_file[0];
size_t name_length = length_of_file_name_and_frills (f);
size_t max_name_length = line_fmt->col_arr[0];

/* Print first entry. */
print_file_name_and_frills (f, 0);

/* Now the rest. */
for (filesno = 1; filesno < cwd_n_used; ++filesno)
{
    size_t col = filesno % cols;

    if (col == 0)
    {

```

```

        putchar (eolbyte);
        pos = 0;
    }
else
{
    indent (pos + name_length, pos + max_name_length);
    pos += max_name_length;
}

f = sorted_file[filesno];
print_file_name_and_frills (f, pos);

name_length = length_of_file_name_and_frills (f);
max_name_length = line_fmt->col_arr[col];
}
putchar (eolbyte);
}

/* Output name + SEP + ' '. */

static void
print_with_separator (char sep)
{
size_t filesno;
size_t pos = 0;

for (filesno = 0; filesno < cwd_n_used; filesno++)
{
    struct fileinfo const *f = sorted_file[filesno];
    size_t len = line_length ? length_of_file_name_and_frills (f) : 0;

    if (filesno != 0)
    {
        char separator;

        if (!line_length
            || ((pos + len + 2 < line_length)
                && (pos <= SIZE_MAX - len - 2)))
        {
            pos += 2;
            separator = ' ';
        }
        else
        {
            pos = 0;
            separator = eolbyte;
        }

        putchar (sep);
        putchar (separator);
    }

    print_file_name_and_frills (f, pos);
    pos += len;
}
putchar (eolbyte);
}

/* Assuming cursor is at position FROM, indent up to position TO.
Use a TAB character instead of two or more spaces whenever possible. */

```

```

static void
indent (size_t from, size_t to)
{
    while (from < to)
    {
        if (tabsize != 0 && to / tabsize > (from + 1) / tabsize)
        {
            putchar ('\t');
            from += tabsize - from % tabsize;
        }
        else
        {
            putchar (' ');
            from++;
        }
    }
}

/* Put DIRNAME/NAME into DEST, handling '.' and '/' properly. */
/* FIXME: maybe remove this function someday. See about using a
non-malloc'ing version of file_name_concat. */

static void
attach (char *dest, char const *dirname, char const *name)
{
    char const *dirnameep = dirname;

    /* Copy dirname if it is not "..". */
    if (dirname[0] != '.' || dirname[1] != 0)
    {
        while (*dirnameep)
            *dest++ = *dirnameep++;
        /* Add '/' if 'dirname' doesn't already end with it. */
        if (dirnameep > dirname && dirnameep[-1] != '/')
            *dest++ = '/';
    }
    while (*name)
        *dest++ = *name++;
    *dest = 0;
}

/* Allocate enough column info suitable for the current number of
files and display columns, and initialize the info to represent the
narrowest possible columns. */

static void
init_column_info (size_t max_cols)
{
    size_t i;

    /* Currently allocated columns in column_info. */
    static size_t column_info_alloc;

    if (column_info_alloc < max_cols)
    {
        size_t new_column_info_alloc;
        size_t *p;

        if (!max_idx || max_cols < max_idx / 2)

```

```

{
/* The number of columns is far less than the display width
   allows. Grow the allocation, but only so that it's
   double the current requirements. If the display is
   extremely wide, this avoids allocating a lot of memory
   that is never needed. */
column_info = xnrealloc (column_info, max_cols,
                        2 * sizeof *column_info);
new_column_info_alloc = 2 * max_cols;
}
else
{
column_info = xnrealloc (column_info, max_idx, sizeof *column_info);
new_column_info_alloc = max_idx;
}

/* Allocate the new size_t objects by computing the triangle
formula n * (n + 1) / 2, except that we don't need to
allocate the part of the triangle that we've already
allocated. Check for address arithmetic overflow. */
{
size_t column_info_growth = new_column_info_alloc - column_info_alloc;
size_t s = column_info_alloc + 1 + new_column_info_alloc;
size_t t = s * column_info_growth;
if (s < new_column_info_alloc || t / column_info_growth != s)
    xalloc_die ();
p = xnmalloc (t / 2, sizeof *p);
}

/* Grow the triangle by parceling out the cells just allocated. */
for (i = column_info_alloc; i < new_column_info_alloc; i++)
{
column_info[i].col_arr = p;
p += i + 1;
}

column_info_alloc = new_column_info_alloc;
}

for (i = 0; i < max_cols; ++i)
{
size_t j;

column_info[i].valid_len = true;
column_info[i].line_len = (i + 1) * MIN_COLUMN_WIDTH;
for (j = 0; j <= i; ++j)
    column_info[i].col_arr[j] = MIN_COLUMN_WIDTH;
}
}

/* Calculate the number of columns needed to represent the current set
of files in the current display width. */

static size_t
calculate_columns (bool by_columns)
{
size_t filesno;          /* Index into cwd_file. */
size_t cols;             /* Number of files across. */

/* Normally the maximum number of columns is determined by the

```

```

screen width. But if few files are available this might limit it
as well. */
size_t max_cols = 0 < max_idx && max_idx < cwd_n_used ? max_idx : cwd_n_used;

init_column_info (max_cols);

/* Compute the maximum number of possible columns. */
for (filesno = 0; filesno < cwd_n_used; ++filesno)
{
    struct fileinfo const *f = sorted_file[filesno];
    size_t name_length = length_of_file_name_and_frills (f);

    for (size_t i = 0; i < max_cols; ++i)
    {
        if (column_info[i].valid_len)
        {
            size_t idx = (by_columns
                          ? filesno / ((cwd_n_used + i) / (i + 1))
                          : filesno % (i + 1));
            size_t real_length = name_length + (idx == i ? 0 : 2);

            if (column_info[i].col_arr[idx] < real_length)
            {
                column_info[i].line_len += (real_length
                                            - column_info[i].col_arr[idx]);
                column_info[i].col_arr[idx] = real_length;
                column_info[i].valid_len = (column_info[i].line_len
                                            < line_length);
            }
        }
    }
}

/* Find maximum allowed columns. */
for (cols = max_cols; 1 < cols; --cols)
{
    if (column_info[cols - 1].valid_len)
        break;
}

return cols;
}

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    printf (_("Usage: %s [OPTION]... [FILE]...\n"), program_name);
    fputs (_("\
List information about the FILEs (the current directory by default).\n\
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.\n\
"), stdout);

    emit_mandatory_arg_note ();

    fputs (_("\
-a, --all           do not ignore entries starting with .\n\

```

```

-A, --almost-all      do not list implied . and ..\n\
--author            with -l, print the author of each file\n\
-b, --escape         print C-style escapes for nongraphic characters\n\
"), stdout);
    fputs (_("\
--block-size=SIZE   with -l, scale sizes by SIZE when printing them;\n\
                     e.g., '--block-size=M'; see SIZE format below\n\
\n\
"), stdout);
    fputs (_("\
-B, --ignore-backups  do not list implied entries ending with ~\n\
"), stdout);
    fputs (_("\
-c                  with -lt: sort by, and show, ctime (time of last\n\
                     change of file status information);\n\
                     with -l: show ctime and sort by name;\n\
                     otherwise: sort by ctime, newest first\n\
\n\
"), stdout);
    fputs (_("\
-C                  list entries by columns\n\
--color[=WHEN]       color the output WHEN; more info below\n\
-d, --directory      list directories themselves, not their contents\n\
-D, --dired          generate output designed for Emacs' dired mode\n\
"), stdout);
    fputs (_("\
-f                  do not sort, enable -aU, disable -ls --color\n\
-F, --classify[=WHEN]  append indicator (one of */=>@!) to entries WHEN\n\
--file-type          likewise, except do not append '*'`n\
"), stdout);
    fputs (_("\
--format=WORD        across -x, commas -m, horizontal -x, long -l,\n\
                     single-column -1, verbose -l, vertical -C\n\
\n\
"), stdout);
    fputs (_("\
--full-time          like -l --time-style=full-iso\n\
"), stdout);
    fputs (_("\
-g                  like -l, but do not list owner\n\
"), stdout);
    fputs (_("\
--group-directories-first\n\
                     group directories before files;\n\
                     can be augmented with a --sort option, but any\n\
                     use of --sort=none (-U) disables grouping\n\
\n\
"), stdout);
    fputs (_("\
-G, --no-group        in a long listing, don't print group names\n\
"), stdout);
    fputs (_("\
-h, --human-readable   with -l and -s, print sizes like 1K 234M 2G etc.\n\
--si                  likewise, but use powers of 1000 not 1024\n\
"), stdout);
    fputs (_("\
-H, --dereference-command-line\n\
                     follow symbolic links listed on the command line\n\
"), stdout);
    fputs (_("\

```

```

--dereference-command-line-symlink-to-dir\n\
    follow each command line symbolic link\n\
    that points to a directory\n\
\n\
"), stdout);
    fputs (_("\
--hide=PATTERN      do not list implied entries matching shell PATTERN\n\
\n\
(overridden by -a or -A)\n\
\n\
"), stdout);
    fputs (_("\
--hyperlink[=WHEN]  hyperlink file names WHEN\n\
"), stdout);
    fputs (_("\
--indicator-style=WORD\n\
            append indicator with style WORD to entry names:\n\
            none (default), slash (-p),\n\
            file-type (--file-type), classify (-F)\n\
\n\
"), stdout);
    fputs (_("\
-i, --inode          print the index number of each file\n\
-l, --ignore=PATTERN do not list implied entries matching shell PATTERN\n\
\n\
"), stdout);
    fputs (_("\
-k, --kibibytes     default to 1024-byte blocks for file system usage;\n\
\n\
used only with -s and per directory totals\n\
\n\
"), stdout);
    fputs (_("\
-l                  use a long listing format\n\
"), stdout);
    fputs (_("\
-L, --dereference   when showing file information for a symbolic\n\
link, show information for the file the link\n\
references rather than for the link itself\n\
\n\
"), stdout);
    fputs (_("\
-m                  fill width with a comma separated list of entries\n\
\n\
"), stdout);
    fputs (_("\
-n, --numeric-uid-gid  like -l, but list numeric user and group IDs\n\
-N, --literal         print entry names without quoting\n\
-o                  like -l, but do not list group information\n\
-p, --indicator-style=slash\n\
            append / indicator to directories\n\
\n\
"), stdout);
    fputs (_("\
-q, --hide-control-chars  print ? instead of nongraphic characters\n\
"), stdout);
    fputs (_("\
--show-control-chars  show nongraphic characters as-is (the default,\n\
unless program is 'ls' and output is a terminal)\n\
\n\
\n\

```

```

"), stdout);
    fputs (_("\
-Q, --quote-name      enclose entry names in double quotes\n\
"), stdout);
    fputs (_("\
--quoting-style=WORD  use quoting style WORD for entry names:\n\
    literal, locale, shell, shell-always,\n\
    shell-escape, shell-escape-always, c, escape\n\
    (overrides QUOTING_STYLE environment variable)\n\
\n\
"), stdout);
    fputs (_("\
-r, --reverse        reverse order while sorting\n\
-R, --recursive      list subdirectories recursively\n\
-s, --size            print the allocated size of each file, in blocks\n\
"), stdout);
    fputs (_("\
-S                  sort by file size, largest first\n\
"), stdout);
    fputs (_("\
--sort=WORD          sort by WORD instead of name: none (-U), size (-S)\n\
,\n\
                time (-t), version (-v), extension (-X), width\n\
\n\
"), stdout);
    fputs (_("\
--time=WORD          select which timestamp used to display or sort;\n\
access time (-u): atime, access, use;\n\
metadata change time (-c): ctime, status;\n\
modified time (default): mtime, modification;\n\
birth time: birth, creation;\n\
with -l, WORD determines which time to show;\n\
with --sort=time, sort by WORD (newest first)\n\
\n\
"), stdout);
    fputs (_("\
--time-style=TIME_STYLE\n\
                time/date format with -l; see TIME_STYLE below\n\
"), stdout);
    fputs (_("\
-t                  sort by time, newest first; see --time\n\
-T, --tabsize=COLS   assume tab stops at each COLS instead of 8\n\
"), stdout);
    fputs (_("\
-u                  with -lt: sort by, and show, access time;\n\
                    with -l: show access time and sort by name;\n\
                    otherwise: sort by access time, newest first\n\
\n\
"), stdout);
    fputs (_("\
-U                  do not sort; list entries in directory order\n\
"), stdout);
    fputs (_("\
-v                  natural sort of (version) numbers within text\n\
"), stdout);
    fputs (_("\
-w, --width=COLS    set output width to COLS.  0 means no limit\n\
-x                  list entries by lines instead of by columns\n\
-X                  sort alphabetically by entry extension\n\
-Z, --context        print any security context of each file\n\

```

```

--zero           end each output line with NUL, not newline\n\
-1              list one file per line\n\
"), stdout);
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
    emit_size_note ();
    fputs (_("\
\n\
The TIME_STYLE argument can be full-iso, long-iso, iso, locale, or +FORMAT.\n\
FORMAT is interpreted like in date(1). If FORMAT is FORMAT1<newline>FORMAT2,\n\
then FORMAT1 applies to non-recent files and FORMAT2 to recent files.\n\
TIME_STYLE prefixed with 'posix-' takes effect only outside the POSIX locale.\n\
Also the TIME_STYLE environment variable sets the default style to use.\n\
"), stdout);
    fputs (_("\
\n\
The WHEN argument defaults to 'always' and can also be 'auto' or 'never'.\n\
"), stdout);
    fputs (_("\
\n\
Using color to distinguish file types is disabled both by default and\n\
with --color=never. With --color=auto, ls emits color codes only when\n\
standard output is connected to a terminal. The LS_COLORS environment\n\
variable can change the settings. Use the dircolors(1) command to set it.\n\
"), stdout);
    fputs (_("\
\n\
Exit status:\n\
0 if OK,\n\
1 if minor problems (e.g., cannot access subdirectory),\n\
2 if serious trouble (e.g., cannot access command-line argument).\n\
"), stdout);
    emit_ancillary_info (PROGRAM_NAME);
}
exit (status);
}
"""

,
"patch":
"""

diff --git a/src/ls.c b/src/ls.c
index f5ac98d..0f5cc72 100644
--- a/src/ls.c
+++ b/src/ls.c
@@ -115,6 +115,38 @@
#include "canonicalize.h"
#include "statx.h"

+#ifdef __MVS__
+#define MVS_LS "/bin/ls"
+#define MVS_LS_LEN (sizeof(MVS_ls)-1)
+#include <unistd.h>
+#include <sys/types.h>
+#include <sys/wait.h>
+
+extern char** environ;
+static bool
+do_mvs_ls (int argc, char **argv)
+{
+
+ pid_t pid = fork();

```

```

+ if (pid == 0) {
+   argv[0] = MVS_LS;
+   execvpe(MVS_LS, argv, environ);
+ } else if (pid > 0) {
+   int wstatus;
+   pid_t waitchild = waitpid(pid, &wstatus, WUNTRACED | WCONTINUED);
+   if (waitchild == -1) {
+     perror("waitpid");
+     return false;
+   }
+   if (WIFEXITED(wstatus)) {
+     return WEXITSTATUS(wstatus) == 0 ? true : false;
+   } else if (WIFSIGNALED(wstatus)) {
+     fprintf(stderr, "killed by signal %d\n", WTERMSIG(wstatus));
+   }
+ }
+}
+endif
+
/* Include <sys/capability.h> last to avoid a clash of <sys/types.h>
   include guards with some premature versions of libcap.
   For more details, see <https://bugzilla.redhat.com/483548>. */
@@ -1926,7 +1958,7 @@ decode_switches (int argc, char **argv)
{
  int oi = -1;
  int c = getopt_long (argc, argv,
-                      "abcdefghijklmnopqrstuvwxyz:ABCDEFGHIJKLMNOPQRSTUVWXYZ:UXZ1",
+                      "abcdefghijklmnopqrstuvwxyz:ABCDEFGHIJKLMNOPQRSTUVWXYZ:UXZ1",
  long_options, &oi);
  if (c == -1)
    break;
@@ -1949,6 +1981,15 @@ decode_switches (int argc, char **argv)
  immediate_dirs = true;
  break;

+ifdef __MVS__
+  case 'E':
+  {
+    int ok = do_mvs_ls(argc, argv);
+    exit (ok ? EXIT_SUCCESS : EXIT_FAILURE);
+  }
+  break;
+endif
+
  case 'f':
  ignore_mode = IGNORE_MINIMAL; /* enable -a */
  sort_opt = sort_none; /* enable -U */
@@ -2110,9 +2151,17 @@ decode_switches (int argc, char **argv)
  break;

  case 'T':
+ifdef __MVS__
+  {
+    int ok = do_mvs_ls(argc, argv);
+    exit (ok ? EXIT_SUCCESS : EXIT_FAILURE);
+  }
+  break;
+else
  tabsize_opt = xnumtoumax (optarg, 0, 0, MIN (PTRDIFF_MAX, SIZE_MAX),
                           "", _("invalid tab size"), LS_FAILURE);

```

```

        break;
+#+endif

    case 'U':
    sort_opt = sort_none;
"""

},

{
    "wrong_code":
"""

/* od -- dump files in octal and other formats
Copyright (C) 1992-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */

/* Written by Jim Meyering. */

#include <config.h>

#include <ctype.h>
#include <float.h>
#include <stdio.h>
#include <getopt.h>
#include <sys/types.h>
#include "system.h"
#include "argmatch.h"
#include "assure.h"
#include "ftoastr.h"
#include "quote.h"
#include "stat-size.h"
#include "xbinary-io.h"
#include "xprintf.h"
#include "xstrtol.h"
#include "xstrtol-error.h"

/* The official name of this program (e.g., no 'g' prefix). */
#define PROGRAM_NAME "od"

#define AUTHORS proper_name ("Jim Meyering")

/* The default number of input bytes per output line. */
#define DEFAULT_BYTES_PER_BLOCK 16

#if HAVE_UNSIGNED_LONG_LONG_INT
typedef unsigned long long int unsigned_long_long_int;
#else
/* This is just a place-holder to avoid a few '#if' directives.
In this case, the type isn't actually used. */

```

```

typedef unsigned long int unsigned_long_long_int;
#endif

#if FLOAT16_SUPPORTED
/* Available since clang 6 (2018), and gcc 7 (2017). */
typedef _Float16 float16;
#else
#define FLOAT16_SUPPORTED 0
/* This is just a place-holder to avoid a few '#if' directives.
   In this case, the type isn't actually used. */
typedef float float16;
#endif

#if BF16_SUPPORTED
/* Available since clang 11 (2020), and gcc 13 (2023). */
typedef __bf16 bfloat16;
#else
#define BF16_SUPPORTED 0
/* This is just a place-holder to avoid a few '#if' directives.
   In this case, the type isn't actually used. */
typedef float bfloat16;
#endif

enum size_spec
{
    NO_SIZE,
    CHAR,
    SHORT,
    INT,
    LONG,
    LONG_LONG,
    /* FIXME: add INTMAX support, too */
    FLOAT_HALF,
    FLOAT_SINGLE,
    FLOAT_DOUBLE,
    FLOAT_LONG_DOUBLE,
    N_SIZE_SPECS
};

enum output_format
{
    SIGNED_DECIMAL,
    UNSIGNED_DECIMAL,
    OCTAL,
    HEXADECIMAL,
    FLOATING_POINT,
    HFLOATING_POINT,
    BFLOATING_POINT,
    NAMED_CHARACTER,
    CHARACTER
};

#define MAX_INTEGRAL_TYPE_SIZE sizeof(unsigned_long_long_int)

/* The maximum number of bytes needed for a format string, including
   the trailing nul. Each format string expects a variable amount of
   padding (guaranteed to be at least 1 plus the field width), then an
   element that will be formatted in the field. */
enum
{

```

```

FMT_BYTES_ALLOCATED =
    (sizeof "%.*99" + 1
     + MAX(sizeof "ld",
           MAX(sizeof "jd",
               MAX(sizeof "jd",
                   MAX(sizeof "ju",
                       sizeof "jx")))))
};

/* Ensure that our choice for FMT_BYTES_ALLOCATED is reasonable. */
static_assert (MAX_INTEGRAL_TYPE_SIZE * CHAR_BIT / 3 <= 99);

/* Each output format specification (from '-t spec' or from
old-style options) is represented by one of these structures. */
struct tspec
{
    enum output_format fmt;
    enum size_spec size; /* Type of input object. */
    /* FIELDS is the number of fields per line, BLANK is the number of
fields to leave blank. WIDTH is width of one field, excluding
leading space, and PAD is total pad to divide among FIELDS.
PAD is at least as large as FIELDS. */
    void (*print_function) (size_t fields, size_t blank, void const *data,
                           char const *fmt, int width, int pad);
    char fmt_string[FMT_BYTES_ALLOCATED]; /* Of the style "%*d". */
    bool hexl_mode_trailer;
    int field_width; /* Minimum width of a field, excluding leading space. */
    int pad_width; /* Total padding to be divided among fields. */
};

/* Convert the number of 8-bit bytes of a binary representation to
the number of characters (digits + sign if the type is signed)
required to represent the same quantity in the specified base/type.
For example, a 32-bit (4-byte) quantity may require a field width
as wide as the following for these types:
11    unsigned octal
11    signed decimal
10    unsigned decimal
8     unsigned hexadecimal */

static char const bytes_to_oct_digits[] =
{0, 3, 6, 8, 11, 14, 16, 19, 22, 25, 27, 30, 32, 35, 38, 41, 43};

static char const bytes_to_signed_dec_digits[] =
{1, 4, 6, 8, 11, 13, 16, 18, 20, 23, 25, 28, 30, 33, 35, 37, 40};

static char const bytes_to_unsigned_dec_digits[] =
{0, 3, 5, 8, 10, 13, 15, 17, 20, 22, 25, 27, 29, 32, 34, 37, 39};

static char const bytes_to_hex_digits[] =
{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32};

/* It'll be a while before we see integral types wider than 16 bytes,
but if/when it happens, this check will catch it. Without this check,
a wider type would provoke a buffer overrun. */
static_assert (MAX_INTEGRAL_TYPE_SIZE
              < ARRAY_CARDINALITY (bytes_to_hex_digits));

/* Make sure the other arrays have the same length. */
static_assert (sizeof bytes_to_oct_digits == sizeof bytes_to_signed_dec_digits);

```

```

static_assert (sizeof bytes_to_oct_digits
              == sizeof bytes_to_unsigned_dec_digits);
static_assert (sizeof bytes_to_oct_digits == sizeof bytes_to_hex_digits);

/* Convert enum size_spec to the size of the named type. */
static const int width_bytes[] =
{
-1,
sizeof (char),
sizeof (short int),
sizeof (int),
sizeof (long int),
sizeof (unsigned_long_long_int),
#if BF16_SUPPORTED
sizeof (bfloat16),
#else
sizeof (float16),
#endif
sizeof (float),
sizeof (double),
sizeof (long double)
};

/* Ensure that for each member of 'enum size_spec' there is an
initializer in the width_bytes array. */
static_assert (ARRAY_CARDINALITY (width_bytes) == N_SIZE_SPECS);

/* Names for some non-printing characters. */
static char const chardata[33][4] =
{
"nul", "soh", "stx", "etx", "eot", "enq", "ack", "bel",
"bs", "ht", "nl", "vt", "ff", "cr", "so", "si",
"dle", "dc1", "dc2", "dc3", "dc4", "nak", "syn", "etb",
"can", "em", "sub", "esc", "fs", "gs", "rs", "us",
"sp"
};

/* Address base (8, 10 or 16). */
static int address_base;

/* The number of octal digits required to represent the largest
address value. */
#define MAX_ADDRESS_LENGTH \
((sizeof (uintmax_t) * CHAR_BIT + CHAR_BIT - 1) / 3)

/* Width of a normal address. */
static int address_pad_len;

/* Minimum length when detecting --strings. */
static size_t string_min;

/* True when in --strings mode. */
static bool flag_dump_strings;

/* True if we should recognize the older non-option arguments
that specified at most one file and optional arguments specifying
offset and pseudo-start address. */
static bool traditional;

/* True if an old-style 'pseudo-address' was specified. */

```

```
static bool flag_pseudo_start;

/* The difference between the old-style pseudo starting address and
the number of bytes to skip. */
static uintmax_t pseudo_offset;

/* Function that accepts an address and an optional following char,
and prints the address and char to stdout. */
static void (*format_address)(uintmax_t, char);

/* The number of input bytes to skip before formatting and writing. */
static uintmax_t n_bytes_to_skip = 0;

/* When false, MAX_BYTGES_TO_FORMAT and END_OFFSET are ignored, and all
input is formatted. */
static bool limit_bytes_to_format = false;

/* The maximum number of bytes that will be formatted. */
static uintmax_t max_bytes_to_format;

/* The offset of the first byte after the last byte to be formatted. */
static uintmax_t end_offset;

/* When true and two or more consecutive blocks are equal, format
only the first block and output an asterisk alone on the following
line to indicate that identical blocks have been elided. */
static bool abbreviate_duplicate_blocks = true;

/* An array of specs describing how to format each input block. */
static struct tspec *spec;

/* The number of format specs. */
static size_t n_specs;

/* The allocated length of SPEC. */
static size_t n_specs_allocated;

/* The number of input bytes formatted per output line. It must be
a multiple of the least common multiple of the sizes associated with
the specified output types. It should be as large as possible, but
no larger than 16 -- unless specified with the -w option. */
static size_t bytes_per_block;

/* Human-readable representation of *file_list (for error messages).
It differs from file_list[-1] only when file_list[-1] is "-". */
static char const *input_filename;

/* A null-terminated list of the file-arguments from the command line. */
static char const *const *file_list;

/* Initializer for file_list if no file-arguments
were specified on the command line. */
static char const *const default_file_list[] = {"-", nullptr};

/* The input stream associated with the current file. */
static FILE *in_stream;

/* If true, at least one of the files we read was standard input. */
static bool have_read_stdin;
```

```

/* Map the size in bytes to a type identifier. */
static enum size_spec integral_type_size[MAX_INTEGRAL_TYPE_SIZE + 1];

#define MAX_FP_TYPE_SIZE sizeof (long double)
static enum size_spec fp_type_size[MAX_FP_TYPE_SIZE + 1];

#ifndef WORDS_BIGENDIAN
# define WORDS_BIGENDIAN 0
#endif

/* Use native endianness by default. */
static bool input_swap;

static char const short_options[] = "A:aBbcDdeFfHhlij:LIN:OoS:st:vw::Xx";

/* For long options that have no equivalent short option, use a
non-character as a pseudo short option, starting with CHAR_MAX + 1. */
enum
{
TRADITIONAL_OPTION = CHAR_MAX + 1,
ENDIAN_OPTION,
};

enum endian_type
{
endian_little,
endian_big
};

static char const *const endian_args[] =
{
"little", "big", nullptr
};

static enum endian_type const endian_types[] =
{
endian_little, endian_big
};

static struct option const long_options[] =
{
{"skip-bytes", required_argument, nullptr, 'j'},
 {"address-radix", required_argument, nullptr, 'A'},
 {"read-bytes", required_argument, nullptr, 'N'},
 {"format", required_argument, nullptr, 't'},
 {"output-duplicates", no_argument, nullptr, 'v'},
 {"strings", optional_argument, nullptr, 'S'},
 {"traditional", no_argument, nullptr, TRADITIONAL_OPTION},
 {"width", optional_argument, nullptr, 'w'},
 {"endian", required_argument, nullptr, ENDIAN_OPTION },
 {GETOPT_HELP_OPTION_DECL},
 {GETOPT_VERSION_OPTION_DECL},
 {nullptr, 0, nullptr, 0}
};

void
usage (int status)
{
if (status != EXIT_SUCCESS)

```

```

    emit_try_help ();
else
{
    printf (_("\
Usage: %s [OPTION]... [FILE]...\n\
or: %s [-abcdfilosx]... [FILE] [[+]OFFSET[.][b]]\n\
or: %s --traditional [OPTION]... [FILE] [[+]OFFSET[.][b]] [+][LABEL][.][b]]\n\
"),
           program_name, program_name, program_name);
    fputs (_("\n\
Write an unambiguous representation, octal bytes by default,\n\
of FILE to standard output. With more than one FILE argument,\n\
concatenate them in the listed order to form the input.\n\
"), stdout);

    emit_stdin_note ();

    fputs (_("\
\n\
If first and second call formats both apply, the second format is assumed\n\
if the last operand begins with + or (if there are 2 operands) a digit.\n\
An OFFSET operand means -j OFFSET. LABEL is the pseudo-address\n\
at first byte printed, incremented when dump is progressing.\n\
For OFFSET and LABEL, a 0x or 0X prefix indicates hexadecimal;\n\
suffixes may be . for octal and b for multiply by 512.\n\
"), stdout);

    emit_mandatory_arg_note ();

    fputs (_("\
-A, --address-radix=RADIX  output format for file offsets; RADIX is one\n\
    of [doxn], for Decimal, Octal, Hex or None\n\
--endian={big|little}  swap input bytes according the specified order\n\
-j, --skip-bytes=BYTES   skip BYTES input bytes first\n\
"), stdout);
    fputs (_("\
-N, --read-bytes=BYTES   limit dump to BYTES input bytes\n\
-S BYTES, --strings[=BYTES] show only NUL terminated strings\n\
    of at least BYTES (3) printable characters\n\
-t, --format=TYPE        select output format or formats\n\
-v, --output-duplicates  do not use * to mark line suppression\n\
-w[BYTES], --width[=BYTES] output BYTES bytes per output line;\n\
    32 is implied when BYTES is not specified\n\
    --traditional          accept arguments in third form above\n\
"), stdout);
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
    fputs (_("\
\n\
\n\
Traditional format specifications may be intermixed; they accumulate:\n\
-a same as -t a, select named characters, ignoring high-order bit\n\
-b same as -t o1, select octal bytes\n\
-c same as -t c, select printable characters or backslash escapes\n\
-d same as -t u2, select unsigned decimal 2-byte units\n\
"), stdout);
    fputs (_("\
-f same as -t fF, select floats\n\
-i same as -t dI, select decimal ints\n\
-l same as -t dL, select decimal longs\n\
"), stdout);
}

```



```

int next_pad = pad * (i - 1) / fields;                                \
int adjusted_width = pad_remaining - next_pad + width;                  \
T x;                                                               \
if (input_swap && sizeof (T) > 1)                                     \
{                                                               \
    size_t j;                                                 \
    union {                                                 \
        T x;                                                 \
        char b[sizeof (T)];                                \
    } u;                                                 \
    for (j = 0; j < sizeof (T); j++)                         \
        u.b[j] = ((char const *) p)[sizeof (T) - 1 - j]; \
    x = u.x;                                              \
}
else
    x = *p;
p++;
ACTION;
pad_remaining = next_pad;
}

#define PRINT_TYPE(N, T)                                         \
PRINT_FIELDS (N, T, char const *fmt_string,                      \
              xprintf (fmt_string, adjusted_width, x))

#define PRINT_FLOATTYPE(N, T, FTOASTR, BUFSIZE)                   \
PRINT_FIELDS (N, T, MAYBE_UNUSED char const *fmt_string,          \
              char buf[BUFSIZE];                                \
              FTOASTR (buf, sizeof buf, 0, 0, x);               \
              xprintf ("%*s", adjusted_width, buf))             \
                                                               \
PRINT_TYPE (print_s_char, signed char)
PRINT_TYPE (print_char, unsigned char)
PRINT_TYPE (print_s_short, short int)
PRINT_TYPE (print_short, unsigned short int)
PRINT_TYPE (print_int, unsigned int)
PRINT_TYPE (print_long, unsigned long int)
PRINT_TYPE (print_long_long, unsigned_long_long_int)

PRINT_FLOATTYPE (print_bfloat, bfloat16, ftoaстр, FLT_BUFSIZE_BOUND)
PRINT_FLOATTYPE (print_halffloat, float16, ftoaстр, FLT_BUFSIZE_BOUND)
PRINT_FLOATTYPE (print_float, float, ftoaстр, FLT_BUFSIZE_BOUND)
PRINT_FLOATTYPE (print_double, double, dtoastr, DBL_BUFSIZE_BOUND)
PRINT_FLOATTYPE (print_long_double, long double, ldtoastr, LDBL_BUFSIZE_BOUND)

#undef PRINT_TYPE
#undef PRINT_FLOATTYPE

static void
dump_hexl_mode_trailer (size_t n_bytes, char const *block)
{
fputs (" >", stdout);
for (size_t i = n_bytes; i > 0; i--)
{
    unsigned char c = *block++;
    unsigned char c2 = (isprint (c) ? c : '.');
    putchar (c2);
}
putchar ('<');

```

```

}

static void
print_named_ascii (size_t fields, size_t blank, void const *block,
    MAYBE_UNUSED char const *unused_fmt_string,
    int width, int pad)
{
unsigned char const *p = block;
uintmax_t i;
int pad_remaining = pad;
for (i = fields; blank < i; i--)
{
    int next_pad = pad * (i - 1) / fields;
    int masked_c = *p++ & 0x7f;
    char const *s;
    char buf[2];

    if (masked_c == 127)
        s = "del";
    else if (masked_c <= 040)
        s = charname[masked_c];
    else
    {
        buf[0] = masked_c;
        buf[1] = 0;
        s = buf;
    }

    xprintf ("%*s", pad_remaining - next_pad + width, s);
    pad_remaining = next_pad;
}
}

static void
print_ascii (size_t fields, size_t blank, void const *block,
    MAYBE_UNUSED char const *unused_fmt_string, int width,
    int pad)
{
unsigned char const *p = block;
uintmax_t i;
int pad_remaining = pad;
for (i = fields; blank < i; i--)
{
    int next_pad = pad * (i - 1) / fields;
    unsigned char c = *p++;
    char const *s;
    char buf[4];

    switch (c)
    {
        case '\0':
            s = "\\\0";
            break;

        case '\a':
            s = "\\\a";
            break;

        case '\b':
            s = "\\\b";
            break;
    }
}
}
```

```

break;

case '\f':
s = "\f";
break;

case '\n':
s = "\n";
break;

case '\r':
s = "\r";
break;

case '\t':
s = "\t";
break;

case '\v':
s = "\v";
break;

default:
sprintf (buf, (isprint (c) ? "%c" : "%03o"), c);
s = buf;
}

xprintf ("%*s", pad_remaining - next_pad + width, s);
pad_remaining = next_pad;
}
}

/* Convert a null-terminated (possibly zero-length) string S to an
int value. If S points to a non-digit set *P to S,
*VAL to 0, and return true. Otherwise, accumulate the integer value of
the string of digits. If the string of digits represents a value
larger than INT_MAX, don't modify *VAL or *P and return false.
Otherwise, advance *P to the first non-digit after S, set *VAL to
the result of the conversion and return true. */

static bool
simple_strtoi (char const *s, char const **p, int *val)
{
int sum;

for (sum = 0; ISDIGIT (*s); s++)
if (ckd_mul (&sum, sum, 10) || ckd_add (&sum, sum, *s - '0'))
return false;
*p = s;
*val = sum;
return true;
}

/* If S points to a single valid modern od format string, put
a description of that format in *TSPEC, make *NEXT point at the
character following the just-decoded format (if *NEXT is non-null),
and return true. If S is not valid, don't modify *NEXT or *TSPEC,
give a diagnostic, and return false. For example, if S were
"d4afL" *NEXT would be set to "afL" and *TSPEC would be
{

```

```

fmt = SIGNED_DECIMAL;
size = INT or LONG; (whichever integral_type_size[4] resolves to)
print_function = print_int; (assuming size == INT)
field_width = 11;
fmt_string = "%*d";
}

pad_width is determined later, but is at least as large as the
number of fields printed per row.
S_ORIG is solely for reporting errors. It should be the full format
string argument.
*/

```

```

static bool ATTRIBUTE_NONNULL ()
decode_one_format (char const *s_orig, char const *s, char const **next,
                   struct tspec *tspec)
{
enum size_spec size_spec;
int size;
enum output_format fmt;
void (*print_function) (size_t, size_t, void const *, char const *,
                       int, int);
char const *p;
char c;
int field_width;

switch (*s)
{
case 'd':
case 'o':
case 'u':
case 'x':
c = *s;
++s;
switch (*s)
{
case 'C':
++s;
size = sizeof (char);
break;

case 'S':
++s;
size = sizeof (short int);
break;

case 'I':
++s;
size = sizeof (int);
break;

case 'L':
++s;
size = sizeof (long int);
break;

default:
if (! simple_strtoi (s, &p, &size))
{
/* The integer at P in S would overflow an int.
A digit string that long is sufficiently odd looking

```

```

        that the following diagnostic is sufficient. */
error (0, 0, _("invalid type string %s"), quote (s_orig));
return false;
}
if (p == s)
    size = sizeof (int);
else
{
if (MAX_INTEGRAL_TYPE_SIZE < size
|| integral_type_size[size] == NO_SIZE)
{
error (0, 0, _("invalid type string %s;\nthis system"
               " doesn't provide a %d-byte integral type"),
       quote (s_orig), size);
return false;
}
s = p;
}
break;
}

#define ISPEC_TO_FORMAT(Spec, Min_format, Long_format, Max_format)      \
((Spec) == LONG_LONG ? (Max_format)                                     \
: ((Spec) == LONG ? (Long_format)                                         \
: (Min_format)))                                                       \
\

size_spec = integral_type_size[size];

switch (c)
{
case 'd':
fmt = SIGNED_DECIMAL;
field_width = bytes_to_signed_dec_digits[size];
sprintf (tspec->fmt_string, "%%" * "%s",
         ISPEC_TO_FORMAT (size_spec, "d", "ld", "jd"));
break;

case 'o':
fmt = OCTAL;
sprintf (tspec->fmt_string, "%%" * ".%d%s",
         (field_width = bytes_to_oct_digits[size]),
         ISPEC_TO_FORMAT (size_spec, "o", "lo", "jo"));
break;

case 'u':
fmt = UNSIGNED_DECIMAL;
field_width = bytes_to_unsigned_dec_digits[size];
sprintf (tspec->fmt_string, "%%" * "%s",
         ISPEC_TO_FORMAT (size_spec, "u", "lu", "ju"));
break;

case 'x':
fmt = HEXADECIMAL;
sprintf (tspec->fmt_string, "%%" * ".%d%s",
         (field_width = bytes_to_hex_digits[size]),
         ISPEC_TO_FORMAT (size_spec, "x", "lx", "jx"));
break;

default:
unreachable ();

```

```

}

switch (size_spec)
{
    case CHAR:
        print_function = (fmt == SIGNED_DECIMAL
            ? print_s_char
            : print_char);
        break;

    case SHORT:
        print_function = (fmt == SIGNED_DECIMAL
            ? print_s_short
            : print_short);
        break;

    case INT:
        print_function = print_int;
        break;

    case LONG:
        print_function = print_long;
        break;

    case LONG_LONG:
        print_function = print_long_long;
        break;

    default:
        affirm (false);
}
break;

case 'f':
fmt = FLOATING_POINT;
++s;
switch (*s)
{
    case 'B':
        ++s;
        fmt = BFLOATING_POINT;
        size = sizeof (bfloat16);
        break;

    case 'H':
        ++s;
        fmt = HFLOATING_POINT;
        size = sizeof (float16);
        break;

    case 'F':
        ++s;
        size = sizeof (float);
        break;

    case 'D':
        ++s;
        size = sizeof (double);
        break;
}

```

```

case 'L':
++s;
size = sizeof (long double);
break;

default:
if (! simple_strtoi (s, &p, &size))
{
/* The integer at P in S would overflow an int.
A digit string that long is sufficiently odd looking
that the following diagnostic is sufficient. */
error (0, 0, _("invalid type string %s"), quote (s_orig));
return false;
}
if (p == s)
size = sizeof (double);
else
{
if (size > MAX_FP_TYPE_SIZE
|| fp_type_size[size] == NO_SIZE
|| (! FLOAT16_SUPPORTED && BF16_SUPPORTED
&& size == sizeof (bfloat16)))
)
{
error (0, 0,
_("invalid type string %s;\n"
"this system doesn't provide a %d-byte"
" floating point type"),
quote (s_orig), size);
return false;
}
s = p;
}
break;
}
size_spec = fp_type_size[size];

if ((! FLOAT16_SUPPORTED && fmt == HFLOATING_POINT)
|| (! BF16_SUPPORTED && fmt == BFLOATING_POINT))
{
error (0, 0,
_("this system doesn't provide a %s floating point type"),
quote (s_orig));
return false;
}

{
struct lconv const *locale = localeconv ();
size_t decimal_point_len =
(locale->decimal_point[0] ? strlen (locale->decimal_point) : 1);

switch (size_spec)
{
case FLOAT_HALF:
print_function = fmt == BFLOATING_POINT
? print_bfloat : print_halffloat;
field_width = FLT_STRLEN_BOUND_L (decimal_point_len);
break;

case FLOAT_SINGLE:

```

```

        print_function = print_float;
        field_width = FLT_STRLEN_BOUND_L (decimal_point_len);
        break;

    case FLOAT_DOUBLE:
        print_function = print_double;
        field_width = DBL_STRLEN_BOUND_L (decimal_point_len);
        break;

    case FLOAT_LONG_DOUBLE:
        print_function = print_long_double;
        field_width = LDBL_STRLEN_BOUND_L (decimal_point_len);
        break;

    default:
        affirm (false);
    }

    break;
}

case 'a':
++$;
fmt = NAMED_CHARACTER;
size_spec = CHAR;
print_function = print_named_ascii;
field_width = 3;
break;

case 'c':
++$;
fmt = CHARACTER;
size_spec = CHAR;
print_function = print_ascii;
field_width = 3;
break;

default:
error (0, 0, _("invalid character '%c' in type string %s"),
      *s, quote (s_orig));
return false;
}

tspec->size = size_spec;
tspec->fmt = fmt;
tspec->print_function = print_function;

tspec->field_width = field_width;
tspec->hexl_mode_trailer = (*s == 'z');
if (tspec->hexl_mode_trailer)
    s++;

*next = s;
return true;
}

/* Given a list of one or more input filenames FILE_LIST, set the global
file pointer IN_STREAM and the global string INPUT_FILENAME to the
first one that can be successfully opened. Modify FILE_LIST to
reference the next filename in the list. A file name of "-" is

```

interpreted as standard input. If any file open fails, give an error message and return false. \*/

```
static bool
open_next_file (void)
{
bool ok = true;

do
{
    input_filename = *file_list;
    if (input_filename == nullptr)
        return ok;
    ++file_list;

    if (STREQ (input_filename, "-"))
    {
        input_filename = _("standard input");
        in_stream = stdin;
        have_read_stdin = true;
        xset_binary_mode (STDIN_FILENO, O_BINARY);
    }
else
{
    in_stream = fopen (input_filename, (O_BINARY ? "rb" : "r"));
    if (in_stream == nullptr)
    {
        error (0, errno, "%s", quotef (input_filename));
        ok = false;
    }
}
while (in_stream == nullptr);

if (limit_bytes_to_format && !flag_dump_strings)
    setvbuf (in_stream, nullptr, _IONBF, 0);

return ok;
}

/* Test whether there have been errors on in_stream, and close it if
it is not standard input. Return false if there has been an error
on in_stream or stdout; return true otherwise. This function will
report more than one error only if both a read and a write error
have occurred. IN_ERRNO, if nonzero, is the error number
corresponding to the most recent action for IN_STREAM. */

static bool
check_and_close (int in_errno)
{
bool ok = true;

if (in_stream != nullptr)
{
    if (!ferror (in_stream))
        in_errno = 0;
    if (STREQ (file_list[-1], "-"))
        clearerr (in_stream);
    else if (fclose (in_stream) != 0 && !in_errno)
        in_errno = errno;
}
```

```

if (in_errno)
{
    error (0, in_errno, "%s", quotef (input_filename));
    ok = false;
}

in_stream = nullptr;
}

if (ferror (stdout))
{
    error (0, 0, _("write error"));
    ok = false;
}

return ok;
}

/* Decode the modern od format string S. Append the decoded
representation to the global array SPEC, reallocating SPEC if
necessary. Return true if S is valid. */

static bool ATTRIBUTE_NONNULL ()
decode_format_string (char const *s)
{
char const *s_orig = s;

while (*s != '\0')
{
    char const *next;

    if (n_specs_allocated <= n_specs)
        spec = X2NREALLOC (spec, &n_specs_allocated);

    if (!decode_one_format (s_orig, s, &next, &spec[n_specs]))
        return false;

    affirm (s != next);
    s = next;
    ++n_specs;
}

return true;
}

/* Given a list of one or more input filenames FILE_LIST, set the global
file pointer IN_STREAM to position N_SKIP in the concatenation of
those files. If any file operation fails or if there are fewer than
N_SKIP bytes in the combined input, give an error message and return
false. When possible, use seek rather than read operations to
advance IN_STREAM. */

static bool
skip (uintmax_t n_skip)
{
bool ok = true;
int in_errno = 0;

if (n_skip == 0)
    return true;

```

```

while (in_stream != nullptr) /* EOF. */
{
    struct stat file_stats;

    /* First try seeking. For large offsets, this extra work is
       worthwhile. If the offset is below some threshold it may be
       more efficient to move the pointer by reading. There are two
       issues when trying to seek:
       - the file must be seekable.
       - before seeking to the specified position, make sure
         that the new position is in the current file.
       Try to do that by getting file's size using fstat.
       But that will work only for regular files. */

    if (fstat (fileno (in_stream), &file_stats) == 0)
    {
        bool usable_size = usable_st_size (&file_stats);

        /* The st_size field is valid for regular files.
           If the number of bytes left to skip is larger than
           the size of the current file, we can decrement n_skip
           and go on to the next file. Skip this optimization also
           when st_size is no greater than the block size, because
           some kernels report nonsense small file sizes for
           proc-like file systems. */
        if (usable_size && STP_BLKSIZEx (&file_stats) < file_stats.st_size)
        {
            if ((uintmax_t) file_stats.st_size < n_skip)
                n_skip -= file_stats.st_size;
            else
            {
                if (fseeko (in_stream, n_skip, SEEK_CUR) != 0)
                {
                    in_errno = errno;
                    ok = false;
                }
                n_skip = 0;
            }
        }

        else if (!usable_size && fseeko (in_stream, n_skip, SEEK_CUR) == 0)
            n_skip = 0;

        /* If it's not a regular file with nonnegative size,
           or if it's so small that it might be in a proc-like file system,
           position the file pointer by reading. */

        else
        {
            char buf[BUFSIZ];
            size_t n_bytes_read, n_bytes_to_read = BUFSIZ;

            while (0 < n_skip)
            {
                if (n_skip < n_bytes_to_read)
                    n_bytes_to_read = n_skip;
                n_bytes_read = fread (buf, 1, n_bytes_to_read, in_stream);
                n_skip -= n_bytes_read;
                if (n_bytes_read != n_bytes_to_read)

```

```

    {
        if (ferror (in_stream))
        {
            in_errno = errno;
            ok = false;
            n_skip = 0;
            break;
        }
        if (feof (in_stream))
            break;
    }
}

if (n_skip == 0)
    break;
}

else /* cannot fstat() file */
{
    error (0, errno, "%s", quotef (input_filename));
    ok = false;
}

ok &= check_and_close (in_errno);

ok &= open_next_file ();
}

if (n_skip != 0)
    error (EXIT_FAILURE, 0, _("cannot skip past end of combined input"));

return ok;
}

static void
format_address_none (MAYBE_UNUSED uintmax_t address,
                     MAYBE_UNUSED char c)
{
}

static void
format_address_std (uintmax_t address, char c)
{
    char buf[MAX_ADDRESS_LENGTH + 2];
    char *p = buf + sizeof buf;
    char const *pbound;

    *--p = '\0';
    *--p = c;
    pbound = p - address_pad_len;

/* Use a special case of the code for each base. This is measurably
   faster than generic code. */
switch (address_base)
{
case 8:
do
    *--p = '0' + (address & 7);
while ((address >= 3) != 0);
}

```

```

break;

case 10:
do
    *--p = '0' + (address % 10);
    while ((address /= 10) != 0);
    break;

case 16:
do
    *--p = "0123456789abcdef"[address & 15];
    while ((address >= 4) != 0);
    break;
}

while (pbound < p)
    *--p = '0';

fputs (p, stdout);
}

static void
format_address_paren (uintmax_t address, char c)
{
putchar ('(');
format_address_std (address, ')');
if (c)
    putchar (c);
}

static void
format_address_label (uintmax_t address, char c)
{
format_address_std (address, ' ');
format_address_paren (address + pseudo_offset, c);
}

/* Write N_BYTES bytes from CURR_BLOCK to standard output once for each
of the N_SPEC format specs. CURRENT_OFFSET is the byte address of
CURR_BLOCK in the concatenation of input files, and it is printed
(optionally) only before the output line associated with the first
format spec. When duplicate blocks are being abbreviated, the output
for a sequence of identical input blocks is the output for the first
block followed by an asterisk alone on a line. It is valid to compare
the blocks PREV_BLOCK and CURR_BLOCK only when N_BYTES ==
BYTES_PER_BLOCK.
That condition may be false only for the last input block. */

static void
write_block (uintmax_t current_offset, size_t n_bytes,
            char const *prev_block, char const *curr_block)
{
static bool first = true;
static bool prev_pair_equal = false;

#define EQUAL_BLOCKS(b1, b2) (memcmp (b1, b2, bytes_per_block) == 0)

if (abbreviate_duplicate_blocks
    && !first && n_bytes == bytes_per_block
    && EQUAL_BLOCKS (prev_block, curr_block))

```

```

{
if (prev_pair_equal)
{
/* The two preceding blocks were equal, and the current
block is the same as the last one, so print nothing. */
}
else
{
printf ("*\n");
prev_pair_equal = true;
}
}
else
{
prev_pair_equal = false;
for (size_t i = 0; i < n_specs; i++)
{
int datum_width = width_bytes[spec[i].size];
int fields_per_block = bytes_per_block / datum_width;
int blank_fields = (bytes_per_block - n_bytes) / datum_width;
if (i == 0)
    format_address (current_offset, '\0');
else
    printf ("%*s", address_pad_len, "");
(*spec[i].print_function) (fields_per_block, blank_fields,
                           curr_block, spec[i].fmt_string,
                           spec[i].field_width, spec[i].pad_width);
if (spec[i].hexl_mode_trailer)
{
/* space-pad out to full line width, then dump the trailer */
int field_width = spec[i].field_width;
int pad_width = (spec[i].pad_width * blank_fields
                 / fields_per_block);
printf ("%*s", blank_fields * field_width + pad_width, "");
dump_hexl_mode_trailer (n_bytes, curr_block);
}
putchar ('\n');
}
}
first = false;
}

/* Read a single byte into *C from the concatenation of the input files
named in the global array FILE_LIST. On the first call to this
function, the global variable IN_STREAM is expected to be an open
stream associated with the input file INPUT_FILENAME. If IN_STREAM
is at end-of-file, close it and update the global variables IN_STREAM
and INPUT_FILENAME so they correspond to the next file in the list.
Then try to read a byte from the newly opened file. Repeat if
necessary until EOF is reached for the last file in FILE_LIST, then
set *C to EOF and return. Subsequent calls do likewise. Return
true if successful. */

static bool
read_char (int *c)
{
bool ok = true;

*c = EOF;

```

```

while (in_stream != nullptr) /* EOF. */
{
    *c = fgetc (in_stream);

    if (*c != EOF)
        break;

    ok &= check_and_close (errno);

    ok &= open_next_file ();
}

return ok;
}

/* Read N bytes into BLOCK from the concatenation of the input files
named in the global array FILE_LIST. On the first call to this
function, the global variable IN_STREAM is expected to be an open
stream associated with the input file INPUT_FILENAME. If all N
bytes cannot be read from IN_STREAM, close IN_STREAM and update
the global variables IN_STREAM and INPUT_FILENAME. Then try to
read the remaining bytes from the newly opened file. Repeat if
necessary until EOF is reached for the last file in FILE_LIST.
On subsequent calls, don't modify BLOCK and return true. Set
*N_BYTGES_IN_BUFFER to the number of bytes read. If an error occurs,
it will be detected through ferror when the stream is about to be
closed. If there is an error, give a message but continue reading
as usual and return false. Otherwise return true. */

static bool
read_block (size_t n, char *block, size_t *n_bytes_in_buffer)
{
bool ok = true;

affirm (0 < n && n <= bytes_per_block);

*n_bytes_in_buffer = 0;

while (in_stream != nullptr) /* EOF. */
{
    size_t n_needed;
    size_t n_read;

    n_needed = n - *n_bytes_in_buffer;
    n_read = fread (block + *n_bytes_in_buffer, 1, n_needed, in_stream);

    *n_bytes_in_buffer += n_read;

    if (n_read == n_needed)
        break;

    ok &= check_and_close (errno);

    ok &= open_next_file ();
}

return ok;
}

/* Return the least common multiple of the sizes associated

```

```

with the format specs. */

ATTRIBUTE PURE
static int
get_lcm (void)
{
int l_c_m = 1;

for (size_t i = 0; i < n_specs; i++)
    l_c_m = lcm (l_c_m, width_bytes[spec[i].size]);
return l_c_m;
}

/* If S is a valid traditional offset specification with an optional
leading '+' return true and set *OFFSET to the offset it denotes. */

static bool
parse_old_offset (char const *s, uintmax_t *offset)
{
int radix;

if (*s == '\0')
    return false;

/* Skip over any leading '+'. */
if (s[0] == '+')
    ++s;

/* Determine the radix we'll use to interpret S. If there is a '.', it's decimal,
otherwise, if the string begins with '0X'or '0x', it's hexadecimal, else octal. */
if (strchr (s, '.') != nullptr)
    radix = 10;
else
{
    if (s[0] == '0' && (s[1] == 'x' || s[1] == 'X'))
        radix = 16;
    else
        radix = 8;
}

return xstrtoumax (s, nullptr, radix, offset, "Bb") == LONGINT_OK;
}

/* Read a chunk of size BYTES_PER_BLOCK from the input files, write the
formatted block to standard output, and repeat until the specified
maximum number of bytes has been read or until all input has been
processed. If the last block read is smaller than BYTES_PER_BLOCK
and its size is not a multiple of the size associated with a format
spec, extend the input block with zero bytes until its length is a
multiple of all format spec sizes. Write the final block. Finally,
write on a line by itself the offset of the byte after the last byte
read. Accumulate return values from calls to read_block and
check_and_close, and if any was false, return false.
Otherwise, return true. */

static bool
dump (void)
{
char *block[2];

```

```

uintmax_t current_offset;
bool idx = false;
bool ok = true;
size_t n_bytes_read;

block[0] = xnmalloc (2, bytes_per_block);
block[1] = block[0] + bytes_per_block;

current_offset = n_bytes_to_skip;

if (limit_bytes_to_format)
{
    while (ok)
    {
        size_t n_needed;
        if (current_offset >= end_offset)
        {
            n_bytes_read = 0;
            break;
        }
        n_needed = MIN (end_offset - current_offset,
                        (uintmax_t) bytes_per_block);
        ok &= read_block (n_needed, block[idx], &n_bytes_read);
        if (n_bytes_read < bytes_per_block)
            break;
        affirm (n_bytes_read == bytes_per_block);
        write_block (current_offset, n_bytes_read,
                     block[!idx], block[idx]);
        if (ferror (stdout))
            ok = false;
        current_offset += n_bytes_read;
        idx = !idx;
    }
}
else
{
    while (ok)
    {
        ok &= read_block (bytes_per_block, block[idx], &n_bytes_read);
        if (n_bytes_read < bytes_per_block)
            break;
        affirm (n_bytes_read == bytes_per_block);
        write_block (current_offset, n_bytes_read,
                     block[!idx], block[idx]);
        if (ferror (stdout))
            ok = false;
        current_offset += n_bytes_read;
        idx = !idx;
    }
}

if (n_bytes_read > 0)
{
    int l_c_m;
    size_t bytes_to_write;

    l_c_m = get_lcm ();
    /* Ensure zero-byte padding up to the smallest multiple of l_c_m that
       is at least as large as n_bytes_read. */
}

```

```

bytes_to_write = l_c_m * ((n_bytes_read + l_c_m - 1) / l_c_m);

memset (block[idx] + n_bytes_read, 0, bytes_to_write - n_bytes_read);
write_block (current_offset, n_bytes_read, block[!idx], block[idx]);
current_offset += n_bytes_read;
}

format_address (current_offset, '\n');

if (limit_bytes_to_format && current_offset >= end_offset)
    ok &= check_and_close (0);

free (block[0]);

return ok;
}

/* STRINGS mode. Find each "string constant" in the input.
A string constant is a run of at least 'string_min' ASCII
graphic (or formatting) characters terminated by a null.
Based on a function written by Richard Stallman for a
traditional version of od. Return true if successful. */

static bool
dump_strings (void)
{
size_t bufsize = MAX (100, string_min);
char *buf = xmalloc (bufsize);
uintmax_t address = n_bytes_to_skip;
bool ok = true;

while (true)
{
    size_t i;
    int c;

    /* See if the next 'string_min' chars are all printing chars. */
    tryline:

    if (limit_bytes_to_format
        && (end_offset < string_min || end_offset - string_min <= address))
        break;

    for (i = 0; i < string_min; i++)
    {
        ok &= read_char (&c);
        address++;
        if (c < 0)
        {
            free (buf);
            return ok;
        }
        if (!isprint (c))
            /* Found a non-printing. Try again starting with next char. */
            goto tryline;
        buf[i] = c;
    }

    /* We found a run of 'string_min' printable characters.
       Now see if it is terminated with a null byte. */
}

```

```

while (!limit_bytes_to_format || address < end_offset)
{
    if (i == bufsize)
    {
        buf = X2REALLOC (buf, &bufsize);
    }
    ok &= read_char (&c);
    address++;
    if (c < 0)
    {
        free (buf);
        return ok;
    }
    if (c == '\0')
        break;          /* It is; print this string. */
    if (! isprint (c))
        goto tryline; /* It isn't; give up on this string. */
    buf[i++] = c;           /* String continues; store it all. */
}

/* If we get here, the string is all printable and null-terminated,
   so print it. It is all in 'buf' and 'i' is its length. */
buf[i] = 0;
format_address (address - i - 1, ' ');

for (i = 0; (c = buf[i]); i++)
{
    switch (c)
    {
        case '\a':
            fputs ("\\a", stdout);
            break;

        case '\b':
            fputs ("\\b", stdout);
            break;

        case '\f':
            fputs ("\\f", stdout);
            break;

        case '\n':
            fputs ("\\n", stdout);
            break;

        case '\r':
            fputs ("\\r", stdout);
            break;

        case '\t':
            fputs ("\\t", stdout);
            break;

        case '\v':
            fputs ("\\v", stdout);
            break;

        default:
            putc (c, stdout);
    }
}

```

```

        }

    putchar ('\n');
}

/* We reach this point only if we search through
(max_bytes_to_format - string_min) bytes before reaching EOF. */

free (buf);

ok &= check_and_close (0);
return ok;
}

int
main (int argc, char **argv)
{
int n_files;
size_t i;
int l_c_m;
idx_t desired_width IF_LINT (= 0);
bool modern = false;
bool width_specified = false;
bool ok = true;
size_t width_per_block = 0;
static char const multipliers[] = "bEGKkMmPQRTYZ0";

/* The old-style 'pseudo starting address' to be printed in parentheses
   after any true address. */
uintmax_t pseudo_start IF_LINT (= 0);

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

atexit (close_stdout);

for (i = 0; i <= MAX_INTEGRAL_TYPE_SIZE; i++)
    integral_type_size[i] = NO_SIZE;

integral_type_size[sizeof (char)] = CHAR;
integral_type_size[sizeof (short int)] = SHORT;
integral_type_size[sizeof (int)] = INT;
integral_type_size[sizeof (long int)] = LONG;
#ifndef HAVE_UNSIGNED_LONG_LONG_INT
/* If 'long int' and 'long long int' have the same size, it's fine
   to overwrite the entry for 'long' with this one. */
integral_type_size[sizeof (unsigned_long_long_int)] = LONG_LONG;
#endif

for (i = 0; i <= MAX_FP_TYPE_SIZE; i++)
    fp_type_size[i] = NO_SIZE;

#ifndef FLOAT16_SUPPORTED
fp_type_size[sizeof (float16)] = FLOAT_HALF;
#endif
#ifndef BF16_SUPPORTED
fp_type_size[sizeof (bfloat16)] = FLOAT_HALF;
#endif
fp_type_size[sizeof (float)] = FLOAT_SINGLE;

```

```

/* The array entry for 'double' is filled in after that for 'long double'
   so that if they are the same size, we avoid any overhead of
   long double computation in libc. */
fp_type_size[sizeof (long double)] = FLOAT_LONG_DOUBLE;
fp_type_size[sizeof (double)] = FLOAT_DOUBLE;

n_specs = 0;
n_specs_allocated = 0;
spec = nullptr;

format_address = format_address_std;
address_base = 8;
address_pad_len = 7;
flag_dump_strings = false;

while (true)
{
    uintmax_t tmp;
    enum strtol_error s_err;
    int oi = -1;
    int c = getopt_long (argc, argv, short_options, long_options, &oi);
    if (c == -1)
        break;

    switch (c)
    {
        case 'A':
            modern = true;
            switch (optarg[0])
            {
                case 'd':
                    format_address = format_address_std;
                    address_base = 10;
                    address_pad_len = 7;
                    break;
                case 'o':
                    format_address = format_address_std;
                    address_base = 8;
                    address_pad_len = 7;
                    break;
                case 'x':
                    format_address = format_address_std;
                    address_base = 16;
                    address_pad_len = 6;
                    break;
                case 'n':
                    format_address = format_address_none;
                    address_pad_len = 0;
                    break;
                default:
                    error (EXIT_FAILURE, 0,
                           ("invalid output address radix '%c':"
                            " it must be one character from [doxn]"),
                           optarg[0]);
                    break;
            }
            break;

        case 'j':
            modern = true;
    }
}

```

```

s_err = xstrtoumax (optarg, nullptr, 0,
                     &n_bytes_to_skip, multipliers);
if (s_err != LONGINT_OK)
    xstrtol_fatal (s_err, oi, c, long_options, optarg);
break;

case 'N':
modern = true;
limit_bytes_to_format = true;

s_err = xstrtoumax (optarg, nullptr, 0, &max_bytes_to_format,
                     multipliers);
if (s_err != LONGINT_OK)
    xstrtol_fatal (s_err, oi, c, long_options, optarg);
break;

case 'S':
modern = true;
if (optarg == nullptr)
    string_min = 3;
else
{
    s_err = xstrtoumax (optarg, nullptr, 0, &tmp, multipliers);
    if (s_err != LONGINT_OK)
        xstrtol_fatal (s_err, oi, c, long_options, optarg);

    /* The minimum string length may be no larger than SIZE_MAX,
       since we may allocate a buffer of this size. */
    if (SIZE_MAX < tmp)
        error (EXIT_FAILURE, 0, _(""%s is too large"), quote (optarg));

    string_min = tmp;
}
flag_dump_strings = true;
break;

case 't':
modern = true;
ok &= decode_format_string (optarg);
break;

case 'v':
modern = true;
abbreviate_duplicate_blocks = false;
break;

case TRADITIONAL_OPTION:
traditional = true;
break;

case ENDIAN_OPTION:
switch (XARGMATCH ("--endian", optarg, endian_args, endian_types))
{
    case endian_big:
        input_swap = ! WORDS_BIGENDIAN;
        break;
    case endian_little:
        input_swap = WORDS_BIGENDIAN;
        break;
}

```

```

break;

/* The next several cases map the traditional format
   specification options to the corresponding modern format
   specs.  GNU od accepts any combination of old- and
   new-style options.  Format specification options accumulate.
   The obsolescent and undocumented formats are compatible
   with FreeBSD 4.10 od. */

#define CASE_OLD_ARG(old_char,new_string)           \
    case old_char:                                \
        ok &= decode_format_string (new_string);    \
        break

CASE_OLD_ARG ('a', "a");
CASE_OLD_ARG ('b', "o1");
CASE_OLD_ARG ('c', "c");
CASE_OLD_ARG ('D', "u4"); /* obsolescent and undocumented */
CASE_OLD_ARG ('d', "u2");
case 'F': /* obsolescent and undocumented alias */
CASE_OLD_ARG ('e', "fD"); /* obsolescent and undocumented */
CASE_OLD_ARG ('f', "fF");
case 'X': /* obsolescent and undocumented alias */
CASE_OLD_ARG ('H', "x4"); /* obsolescent and undocumented */
CASE_OLD_ARG ('i', "dl");
case 'I': case 'L': /* obsolescent and undocumented aliases */
CASE_OLD_ARG ('l', "dL");
CASE_OLD_ARG ('O', "o4"); /* obsolescent and undocumented */
case 'B': /* obsolescent and undocumented alias */
CASE_OLD_ARG ('o', "o2");
CASE_OLD_ARG ('s', "d2");
case 'h': /* obsolescent and undocumented alias */
CASE_OLD_ARG ('x', "x2");

#undef CASE_OLD_ARG

case 'w':
modern = true;
width_specified = true;
if (optarg == nullptr)
{
    desired_width = 32;
}
else
{
    intmax_t w_tmp;
    s_err = xstrtoimax (optarg, nullptr, 10, &w_tmp, "");
    if (s_err != LONGINT_OK || w_tmp <= 0)
        xstrtol_fatal (s_err, oi, c, long_options, optarg);
    if (ckd_add (&desired_width, w_tmp, 0))
        error (EXIT_FAILURE, 0, _("'%s is too large"), quote (optarg));
}
break;

case_GETOPT_HELP_CHAR;

case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

default:
usage (EXIT_FAILURE);

```

```

        break;
    }
}

if (!ok)
    return EXIT_FAILURE;

if (flag_dump_strings && n_specs > 0)
    error (EXIT_FAILURE, 0,
           _("no type may be specified when dumping strings"));

n_files = argc - optind;

/* If the --traditional option is used, there may be from
   0 to 3 remaining command line arguments; handle each case
   separately.
   od [file] [[+]offset[.][]b] [[+]label[.][]b]]
The offset and label have the same syntax.

If --traditional is not given, and if no modern options are
given, and if the offset begins with + or (if there are two
operands) a digit, accept only this form, as per POSIX:
   od [file] [[+]offset[.][]b]]
*/
if (!modern || traditional)
{
    uintmax_t o1;
    uintmax_t o2;

    switch (n_files)
    {
        case 1:
            if ((traditional || argv[optind][0] == '+')
                && parse_old_offset (argv[optind], &o1))
            {
                n_bytes_to_skip = o1;
                --n_files;
                ++argv;
            }
            break;

        case 2:
            if ((traditional || argv[optind + 1][0] == '+'
                || ISDIGIT (argv[optind + 1][0]))
                && parse_old_offset (argv[optind + 1], &o2))
            {
                if (traditional && parse_old_offset (argv[optind], &o1))
                {
                    n_bytes_to_skip = o1;
                    flag_pseudo_start = true;
                    pseudo_start = o2;
                    argv += 2;
                    n_files -= 2;
                }
            }
            else
            {
                n_bytes_to_skip = o2;
                --n_files;
                argv[optind + 1] = argv[optind];
            }
    }
}

```

```

        ++argv;
    }
}

break;

case 3:
if (traditional
    && parse_old_offset (argv[optind + 1], &o1)
    && parse_old_offset (argv[optind + 2], &o2))
{
    n_bytes_to_skip = o1;
    flag_pseudo_start = true;
    pseudo_start = o2;
    argv[optind + 2] = argv[optind];
    argv += 2;
    n_files -= 2;
}
break;
}

if (traditional && 1 < n_files)
{
    error (0, 0, _("extra operand %s"), quote (argv[optind + 1]));
    error (0, 0, "%s",
           _("compatibility mode supports at most one file"));
    usage (EXIT_FAILURE);
}
}

if (flag_pseudo_start)
{
    if (format_address == format_address_none)
    {
        address_base = 8;
        address_pad_len = 7;
        format_address = format_address_paren;
    }
    else
        format_address = format_address_label;
}

if (limit_bytes_to_format)
{
    end_offset = n_bytes_to_skip + max_bytes_to_format;
    if (end_offset < n_bytes_to_skip)
        error (EXIT_FAILURE, 0, _("skip-bytes + read-bytes is too large"));
}

if (n_specs == 0)
    decode_format_string ("oS");

if (n_files > 0)
{
    /* Set the global pointer FILE_LIST so that it
       references the first file-argument on the command-line. */

    file_list = (char const *const *) &argv[optind];
}
else
{

```

```

/* No files were listed on the command line.
   Set the global pointer FILE_LIST so that it
   references the null-terminated list of one name: "-". */

file_list = default_file_list;
}

/* open the first input file */
ok = open_next_file ();
if (in_stream == nullptr)
    goto cleanup;

/* skip over any unwanted header bytes */
ok &= skip (n_bytes_to_skip);
if (in_stream == nullptr)
    goto cleanup;

pseudo_offset = (flag_pseudo_start ? pseudo_start - n_bytes_to_skip : 0);

/* Compute output block length. */
l_c_m = get_lcm ();

if (width_specified)
{
    if (desired_width != 0 && desired_width % l_c_m == 0)
        bytes_per_block = desired_width;
    else
    {
        error (0, 0, _("warning: invalid width %td; using %d instead"),
               desired_width, l_c_m);
        bytes_per_block = l_c_m;
    }
}
else
{
    if (l_c_m < DEFAULT_BYTES_PER_BLOCK)
        bytes_per_block = l_c_m * (DEFAULT_BYTES_PER_BLOCK / l_c_m);
    else
        bytes_per_block = l_c_m;
}

/* Compute padding necessary to align output block. */
for (i = 0; i < n_specs; i++)
{
    int fields_per_block = bytes_per_block / width_bytes[spec[i].size];
    int block_width = (spec[i].field_width + 1) * fields_per_block;
    if (width_per_block < block_width)
        width_per_block = block_width;
}
for (i = 0; i < n_specs; i++)
{
    int fields_per_block = bytes_per_block / width_bytes[spec[i].size];
    int block_width = spec[i].field_width * fields_per_block;
    spec[i].pad_width = width_per_block - block_width;
}

#ifndef DEBUG
printf ("lcm=%d, width_per_block=%zu\n", l_c_m, width_per_block);
#endif

for (i = 0; i < n_specs; i++)
{

```

```

int fields_per_block = bytes_per_block / width_bytes[spec[i].size];
affirm (bytes_per_block % width_bytes[spec[i].size] == 0);
affirm (1 <= spec[i].pad_width / fields_per_block);
printf ("%d: fmt=%\"s\" in_width=%d out_width=%d pad=%d\n",
       i, spec[i].fmt_string, width_bytes[spec[i].size],
       spec[i].field_width, spec[i].pad_width);
}
#endif

ok &= (flag_dump_strings ? dump_strings () : dump ());

cleanup:

if (have_read_stdin && fclose (stdin) == EOF)
    error (EXIT_FAILURE, errno, _("standard input"));

return ok ? EXIT_SUCCESS : EXIT_FAILURE;
}

"""
"error_category":"File Handling and Input Stream Management",
"error": " EDC5126I: Input stream conversion failed on z/OS, caused by using
_disableautocvt() on standard input without proper context.",
"correct_code":
"""

/* od -- dump files in octal and other formats
Copyright (C) 1992-2024 Free Software Foundation, Inc.


```

This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License  
along with this program. If not, see <<https://www.gnu.org/licenses/>>. \*/

/\* Written by Jim Meyering. \*/

```

#include <config.h>

#include <ctype.h>
#include <float.h>
#include <stdio.h>
#include <getopt.h>
#include <sys/types.h>
#include "system.h"
#include "argmatch.h"
#include "assure.h"
#include "ftoaстр.h"
#include "quote.h"
#include "stat-size.h"
#include "xbinary-io.h"
#include "xprintf.h"
#include "xstrtol.h"
#include "xstrtol-error.h"

```

```

/* The official name of this program (e.g., no 'g' prefix). */
#define PROGRAM_NAME "od"

#define AUTHORS proper_name ("Jim Meyering")

/* The default number of input bytes per output line. */
#define DEFAULT_BYTES_PER_BLOCK 16

#if HAVE_UNSIGNED_LONG_LONG_INT
typedef unsigned long long int unsigned_long_long_int;
#else
/* This is just a place-holder to avoid a few '#if' directives.
In this case, the type isn't actually used. */
typedef unsigned long int unsigned_long_long_int;
#endif

#if FLOAT16_SUPPORTED
/* Available since clang 6 (2018), and gcc 7 (2017). */
typedef __Float16 float16;
#else
# define FLOAT16_SUPPORTED 0
/* This is just a place-holder to avoid a few '#if' directives.
In this case, the type isn't actually used. */
typedef float float16;
#endif

#if BF16_SUPPORTED
/* Available since clang 11 (2020), and gcc 13 (2023). */
typedef __bf16 bfloat16;
#else
# define BF16_SUPPORTED 0
/* This is just a place-holder to avoid a few '#if' directives.
In this case, the type isn't actually used. */
typedef float bfloat16;
#endif

enum size_spec
{
    NO_SIZE,
    CHAR,
    SHORT,
    INT,
    LONG,
    LONG_LONG,
    /* FIXME: add INTMAX support, too */
    FLOAT_HALF,
    FLOAT_SINGLE,
    FLOAT_DOUBLE,
    FLOAT_LONG_DOUBLE,
    N_SIZE_SPECS
};

enum output_format
{
    SIGNED_DECIMAL,
    UNSIGNED_DECIMAL,
    OCTAL,
    HEXADECIMAL,
    FLOATING_POINT,
    HFLOATING_POINT,

```

```

BFLOATING_POINT,
NAMED_CHARACTER,
CHARACTER
};

#define MAX_INTEGRAL_TYPE_SIZE sizeof (unsigned_long_long_int)

/* The maximum number of bytes needed for a format string, including
the trailing nul. Each format string expects a variable amount of
padding (guaranteed to be at least 1 plus the field width), then an
element that will be formatted in the field. */
enum
{
    FMT_BYTES_ALLOCATED =
        (sizeof "%.*99" + 1
         + MAX (sizeof "ld",
                 MAX (sizeof "jd",
                         MAX (sizeof "jd",
                                 MAX (sizeof "ju",
                                         sizeof "jx")))))
};

/* Ensure that our choice for FMT_BYTES_ALLOCATED is reasonable. */
static_assert (MAX_INTEGRAL_TYPE_SIZE * CHAR_BIT / 3 <= 99);

/* Each output format specification (from '-t spec' or from
old-style options) is represented by one of these structures. */
struct tspec
{
    enum output_format fmt;
    enum size_spec size; /* Type of input object. */
    /* FIELDS is the number of fields per line, BLANK is the number of
fields to leave blank. WIDTH is width of one field, excluding
leading space, and PAD is total pad to divide among FIELDS.
PAD is at least as large as FIELDS. */
    void (*print_function) (size_t fields, size_t blank, void const *data,
                           char const *fmt, int width, int pad);
    char fmt_string[FMT_BYTES_ALLOCATED]; /* Of the style "%*d". */
    bool hexl_mode_trailer;
    int field_width; /* Minimum width of a field, excluding leading space. */
    int pad_width; /* Total padding to be divided among fields. */
};

/* Convert the number of 8-bit bytes of a binary representation to
the number of characters (digits + sign if the type is signed)
required to represent the same quantity in the specified base/type.
For example, a 32-bit (4-byte) quantity may require a field width
as wide as the following for these types:
11    unsigned octal
11    signed decimal
10    unsigned decimal
8     unsigned hexadecimal */

static char const bytes_to_oct_digits[] =
{0, 3, 6, 8, 11, 14, 16, 19, 22, 25, 27, 30, 32, 35, 38, 41, 43};

static char const bytes_to_signed_dec_digits[] =
{1, 4, 6, 8, 11, 13, 16, 18, 20, 23, 25, 28, 30, 33, 35, 37, 40};

static char const bytes_to_unsigned_dec_digits[] =

```

```

{0, 3, 5, 8, 10, 13, 15, 17, 20, 22, 25, 27, 29, 32, 34, 37, 39};

static char const bytes_to_hex_digits[] =
{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32};

/* It'll be a while before we see integral types wider than 16 bytes,
but if/when it happens, this check will catch it. Without this check,
a wider type would provoke a buffer overrun. */
static_assert (MAX_INTEGRAL_TYPE_SIZE
    < ARRAY_CARDINALITY (bytes_to_hex_digits));

/* Make sure the other arrays have the same length. */
static_assert (sizeof bytes_to_oct_digits == sizeof bytes_to_signed_dec_digits);
static_assert (sizeof bytes_to_oct_digits
    == sizeof bytes_to_unsigned_dec_digits);
static_assert (sizeof bytes_to_oct_digits == sizeof bytes_to_hex_digits);

/* Convert enum size_spec to the size of the named type. */
static const int width_bytes[] =
{
-1,
sizeof (char),
sizeof (short int),
sizeof (int),
sizeof (long int),
sizeof (unsigned_long_long_int),
#if BF16_SUPPORTED
sizeof (bfloat16),
#else
sizeof (float16),
#endif
sizeof (float),
sizeof (double),
sizeof (long double)
};

/* Ensure that for each member of 'enum size_spec' there is an
initializer in the width_bytes array. */
static_assert (ARRAY_CARDINALITY (width_bytes) == N_SIZE_SPECS);

/* Names for some non-printing characters. */
static char const charname[33][4] =
{
"nul", "soh", "stx", "etx", "eot", "enq", "ack", "bel",
"bs", "ht", "nl", "vt", "ff", "cr", "so", "si",
"dle", "dc1", "dc2", "dc3", "dc4", "nak", "syn", "etb",
"can", "em", "sub", "esc", "fs", "gs", "rs", "us",
"sp"
};

/* Address base (8, 10 or 16). */
static int address_base;

/* The number of octal digits required to represent the largest
address value. */
#define MAX_ADDRESS_LENGTH \
((sizeof (uintmax_t) * CHAR_BIT + CHAR_BIT - 1) / 3)

/* Width of a normal address. */
static int address_pad_len;

```

```

/* Minimum length when detecting --strings. */
static size_t string_min;

/* True when in --strings mode. */
static bool flag_dump_strings;

/* True if we should recognize the older non-option arguments
   that specified at most one file and optional arguments specifying
   offset and pseudo-start address. */
static bool traditional;

/* True if an old-style 'pseudo-address' was specified. */
static bool flag_pseudo_start;

/* The difference between the old-style pseudo starting address and
   the number of bytes to skip. */
static uintmax_t pseudo_offset;

/* Function that accepts an address and an optional following char,
   and prints the address and char to stdout. */
static void (*format_address) (uintmax_t, char);

/* The number of input bytes to skip before formatting and writing. */
static uintmax_t n_bytes_to_skip = 0;

/* When false, MAX_BYTES_TO_FORMAT and END_OFFSET are ignored, and all
   input is formatted. */
static bool limit_bytes_to_format = false;

/* The maximum number of bytes that will be formatted. */
static uintmax_t max_bytes_to_format;

/* The offset of the first byte after the last byte to be formatted. */
static uintmax_t end_offset;

/* When true and two or more consecutive blocks are equal, format
   only the first block and output an asterisk alone on the following
   line to indicate that identical blocks have been elided. */
static bool abbreviate_duplicate_blocks = true;

/* An array of specs describing how to format each input block. */
static struct tspec *spec;

/* The number of format specs. */
static size_t n_specs;

/* The allocated length of SPEC. */
static size_t n_specs_allocated;

/* The number of input bytes formatted per output line. It must be
   a multiple of the least common multiple of the sizes associated with
   the specified output types. It should be as large as possible, but
   no larger than 16 -- unless specified with the -w option. */
static size_t bytes_per_block;

/* Human-readable representation of *file_list (for error messages).
   It differs from file_list[-1] only when file_list[-1] is "-". */
static char const *input_filename;

```

```

/* A null-terminated list of the file-arguments from the command line. */
static char const *const *file_list;

/* Initializer for file_list if no file-arguments
   were specified on the command line. */
static char const *const default_file_list[] = {"-", nullptr};

/* The input stream associated with the current file. */
static FILE *in_stream;

/* If true, at least one of the files we read was standard input. */
static bool have_read_stdin;

/* Map the size in bytes to a type identifier. */
static enum size_spec integral_type_size[MAX_INTEGRAL_TYPE_SIZE + 1];

#define MAX_FP_TYPE_SIZE sizeof (long double)
static enum size_spec fp_type_size[MAX_FP_TYPE_SIZE + 1];

#ifndef WORDS_BIGENDIAN
#define WORDS_BIGENDIAN 0
#endif

/* Use native endianness by default. */
static bool input_swap;

static char const short_options[] = "A:aBbcDdeFfHhlj:LIN:OoS:st:vw::Xx";

/* For long options that have no equivalent short option, use a
   non-character as a pseudo short option, starting with CHAR_MAX + 1. */
enum
{
    TRADITIONAL_OPTION = CHAR_MAX + 1,
    ENDIAN_OPTION,
};

enum endian_type
{
    endian_little,
    endian_big
};

static char const *const endian_args[] =
{
    "little", "big", nullptr
};

static enum endian_type const endian_types[] =
{
    endian_little, endian_big
};

static struct option const long_options[] =
{
    {"skip-bytes", required_argument, nullptr, 'j'},
    {"address-radix", required_argument, nullptr, 'A'},
    {"read-bytes", required_argument, nullptr, 'N'},
    {"format", required_argument, nullptr, 't'},
    {"output-duplicates", no_argument, nullptr, 'v'},
    {"strings", optional_argument, nullptr, 'S'},
};

```

```

{ "traditional", no_argument, nullptr, TRADITIONAL_OPTION },
{ "width", optional_argument, nullptr, 'w' },
{ "Endian", required_argument, nullptr, ENDIAN_OPTION },

{GETOPT_HELP_OPTION_DECL},
{GETOPT_VERSION_OPTION_DECL},
{nullptr, 0, nullptr, 0}
};

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    {
        printf (_("\
Usage: %s [OPTION]... [FILE]...\n\
or: %s [-abcdfilosx]... [FILE] [[+]OFFSET[.]][b]]\n\
or: %s --traditional [OPTION]... [FILE] [[+]OFFSET[.]][b] [+][LABEL][.]][b]]\n\
"),
            program_name, program_name, program_name);
        fputs (_("\n\
Write an unambiguous representation, octal bytes by default,\n\
of FILE to standard output. With more than one FILE argument,\n\
concatenate them in the listed order to form the input.\n\
"), stdout);

    emit_stdin_note ();

    fputs (_("\
\n\
If first and second call formats both apply, the second format is assumed\n\
if the last operand begins with + or (if there are 2 operands) a digit.\n\
An OFFSET operand means -j OFFSET. LABEL is the pseudo-address\n\
at first byte printed, incremented when dump is progressing.\n\
For OFFSET and LABEL, a 0x or 0X prefix indicates hexadecimal;\n\
suffixes may be . for octal and b for multiply by 512.\n\
"), stdout);

    emit_mandatory_arg_note ();

    fputs (_("\
-A, --address-radix=RADIX  output format for file offsets; RADIX is one\n\
    of [doxn], for Decimal, Octal, Hex or None\n\
--endian={big|little}  swap input bytes according the specified order\n\
-j, --skip-bytes=BYTES  skip BYTES input bytes first\n\
"), stdout);
    fputs (_("\
-N, --read-bytes=BYTES  limit dump to BYTES input bytes\n\
-S BYTES, --strings[=BYTES]  show only NUL terminated strings\n\
    of at least BYTES (3) printable characters\n\
-t, --format=TYPE  select output format or formats\n\
-v, --output-duplicates  do not use * to mark line suppression\n\
-w[BYTES], --width[=BYTES]  output BYTES bytes per output line;\n\
    32 is implied when BYTES is not specified\n\
--traditional  accept arguments in third form above\n\
"), stdout);
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
}

```

```

        fputs (_("\
\n\
\n\
Traditional format specifications may be intermixed; they accumulate:\n\
-a same as -t a, select named characters, ignoring high-order bit\n\
-b same as -t o1, select octal bytes\n\
-c same as -t c, select printable characters or backslash escapes\n\
-d same as -t u2, select unsigned decimal 2-byte units\n\
"), stdout);
        fputs (_("\
-f same as -t fF, select floats\n\
-i same as -t dl, select decimal ints\n\
-l same as -t dL, select decimal longs\n\
-o same as -t o2, select octal 2-byte units\n\
-s same as -t d2, select decimal 2-byte units\n\
-x same as -t x2, select hexadecimal 2-byte units\n\
"), stdout);
        fputs (_("\
\n\
\n\
TYPE is made up of one or more of these specifications:\n\
a named character, ignoring high-order bit\n\
c printable character or backslash escape\n\
"), stdout);
        fputs (_("\
d[SIZE] signed decimal, SIZE bytes per integer\n\
f[SIZE] floating point, SIZE bytes per float\n\
o[SIZE] octal, SIZE bytes per integer\n\
u[SIZE] unsigned decimal, SIZE bytes per integer\n\
x[SIZE] hexadecimal, SIZE bytes per integer\n\
"), stdout);
        fputs (_("\
\n\
SIZE is a number. For TYPE in [doux], SIZE may also be C for\n\
sizeof(char), S for sizeof(short), I for sizeof(int) or L for\n\
sizeof(long). If TYPE is f, SIZE may also be B for Brain 16 bit,\n\
H for Half precision float, F for sizeof(float), D for sizeof(double),\n\
or L for sizeof(long double).\n\
"), stdout);
        fputs (_("\
\n\
Adding a z suffix to any type displays printable characters at the end of\n\
each output line.\n\
"), stdout);
        fputs (_("\
\n\
\n\
BYTES is hex with 0x or 0X prefix, and may have a multiplier suffix:\n\
b 512\n\
KB 1000\n\
K 1024\n\
MB 1000*1000\n\
M 1024*1024\n\
and so on for G, T, P, E, Z, Y, R, Q.\n\
Binary prefixes can be used, too: KiB=K, MiB=M, and so on.\n\
"), stdout);
        emit_ancillary_info (PROGRAM_NAME);
    }
exit (status);
}

```

```

/* Define the print functions. */

#define PRINT_FIELDS(N, T, FMT_STRING_DECL, ACTION)           \
static void                                                 \
N (size_t fields, size_t blank, void const *block,          \
FMT_STRING_DECL, int width, int pad)                         \
{                                                       \
T const *p = block;                                         \
uintmax_t i;                                                 \
int pad_remaining = pad;                                     \
for (i = fields; blank < i; i--)                            \
{                                                       \
    int next_pad = pad * (i - 1) / fields;                  \
    int adjusted_width = pad_remaining - next_pad + width; \
    T x;                                                 \
    if (input_swap && sizeof (T) > 1)                      \
    {                                                       \
        size_t j;                                           \
        union {                                              \
            T x;                                             \
            char b[sizeof (T)];                                \
        } u;                                               \
        for (j = 0; j < sizeof (T); j++)                     \
            u.b[j] = ((char const *) p)[sizeof (T) - 1 - j]; \
        x = u.x;                                           \
    }                                                       \
    else                                                   \
        x = *p;                                           \
    p++;                                                 \
    ACTION;                                              \
    pad_remaining = next_pad;                               \
}                                                       \
}

#define PRINT_TYPE(N, T)                                     \
PRINT_FIELDS (N, T, char const *fmt_string,                \
              xprintf (fmt_string, adjusted_width, x))

#define PRINT_FLOATTYPE(N, T, FTOASTR, BUFSIZE)           \
PRINT_FIELDS (N, T, MAYBE_UNUSED char const *fmt_string, \
              char buf[BUFSIZE];                           \
              FTOASTR (buf, sizeof buf, 0, 0, x);          \
              xprintf ("%*s", adjusted_width, buf))

PRINT_TYPE (print_s_char, signed char)
PRINT_TYPE (print_char, unsigned char)
PRINT_TYPE (print_s_short, short int)
PRINT_TYPE (print_short, unsigned short int)
PRINT_TYPE (print_int, unsigned int)
PRINT_TYPE (print_long, unsigned long int)
PRINT_TYPE (print_long_long, unsigned long long int)

PRINT_FLOATTYPE (print_bfloat, bfloat16, ftoastr, FLT_BUFSIZE_BOUND)
PRINT_FLOATTYPE (print_halffloat, float16, ftoastr, FLT_BUFSIZE_BOUND)
PRINT_FLOATTYPE (print_float, float, ftoastr, FLT_BUFSIZE_BOUND)
PRINT_FLOATTYPE (print_double, double, dtoastr, DBL_BUFSIZE_BOUND)
PRINT_FLOATTYPE (print_long_double, long double, ldoastr, LDBL_BUFSIZE_BOUND)

#endif /* !defined (PRINT_TYPE) */

```

```

#define PRINT_FLOATTYPE

static void
dump_hexl_mode_trailer (size_t n_bytes, char const *block)
{
    fputs (" >", stdout);
    for (size_t i = n_bytes; i > 0; i--)
    {
        unsigned char c = *block++;
        unsigned char c2 = (isprint (c) ? c : '.');
        putchar (c2);
    }
    putchar ('<');
}

static void
print_named_ascii (size_t fields, size_t blank, void const *block,
    MAYBE_UNUSED char const *unused_fmt_string,
    int width, int pad)
{
    unsigned char const *p = block;
    uintmax_t i;
    int pad_remaining = pad;
    for (i = fields; blank < i; i--)
    {
        int next_pad = pad * (i - 1) / fields;
        int masked_c = *p++ & 0x7f;
        char const *s;
        char buf[2];

        if (masked_c == 127)
            s = "del";
        else if (masked_c <= 040)
            s = charname[masked_c];
        else
        {
            buf[0] = masked_c;
            buf[1] = 0;
            s = buf;
        }

        xprintf ("%*s", pad_remaining - next_pad + width, s);
        pad_remaining = next_pad;
    }
}

static void
print_ascii (size_t fields, size_t blank, void const *block,
    MAYBE_UNUSED char const *unused_fmt_string, int width,
    int pad)
{
    unsigned char const *p = block;
    uintmax_t i;
    int pad_remaining = pad;
    for (i = fields; blank < i; i--)
    {
        int next_pad = pad * (i - 1) / fields;
        unsigned char c = *p++;
        char const *s;
        char buf[4];

```

```

switch (c)
{
    case '\0':
        s = "\\0";
        break;

    case '\a':
        s = "\\a";
        break;

    case '\b':
        s = "\\b";
        break;

    case '\f':
        s = "\\f";
        break;

    case '\n':
        s = "\\n";
        break;

    case '\r':
        s = "\\r";
        break;

    case '\t':
        s = "\\t";
        break;

    case '\v':
        s = "\\v";
        break;

    default:
        sprintf (buf, (isprint (c) ? "%c" : "%03o"), c);
        s = buf;
}

xprintf ("%*s", pad_remaining - next_pad + width, s);
pad_remaining = next_pad;
}

/* Convert a null-terminated (possibly zero-length) string S to an
int value. If S points to a non-digit set *P to S,
*VAL to 0, and return true. Otherwise, accumulate the integer value of
the string of digits. If the string of digits represents a value
larger than INT_MAX, don't modify *VAL or *P and return false.
Otherwise, advance *P to the first non-digit after S, set *VAL to
the result of the conversion and return true. */
static bool
simple_strtoi (char const *s, char const **p, int *val)
{
    int sum;

    for (sum = 0; ISDIGIT (*s); s++)
        if (ckd_mul (&sum, sum, 10) || ckd_add (&sum, sum, *s - '0'))

```

```

        return false;
    *p = s;
    *val = sum;
    return true;
}

/* If S points to a single valid modern od format string, put
a description of that format in *TSPEC, make *NEXT point at the
character following the just-decoded format (if *NEXT is non-null),
and return true. If S is not valid, don't modify *NEXT or *TSPEC,
give a diagnostic, and return false. For example, if S were
"d4afL" *NEXT would be set to "afL" and *TSPEC would be
{

```

```

    fmt = SIGNED_DECIMAL;
    size = INT or LONG; (whichever integral_type_size[4] resolves to)
    print_function = print_int; (assuming size == INT)
    field_width = 11;
    fmt_string = "%*d";
}

```

pad\_width is determined later, but is at least as large as the number of fields printed per row.

S\_ORIG is solely for reporting errors. It should be the full format string argument.

\*/

```

static bool ATTRIBUTE_NONNULL ()
decode_one_format (char const *s_orig, char const *s, char const **next,
                  struct tspec *tspec)
{
enum size_spec size_spec;
int size;
enum output_format fmt;
void (*print_function) (size_t, size_t, void const *, char const *,
                       int, int);
char const *p;
char c;
int field_width;

switch (*s)
{
case 'd':
case 'o':
case 'u':
case 'x':
c = *s;
++s;
switch (*s)
{
case 'C':
++s;
size = sizeof (char);
break;

case 'S':
++s;
size = sizeof (short int);
break;

case 'I':
++s;

```

```

size = sizeof (int);
break;

case 'L':
++s;
size = sizeof (long int);
break;

default:
if (! simple_strtoi (s, &p, &size))
{
/* The integer at P in S would overflow an int.
A digit string that long is sufficiently odd looking
that the following diagnostic is sufficient. */
error (0, 0, _("invalid type string %s"), quote (s_orig));
return false;
}
if (p == s)
size = sizeof (int);
else
{
if (MAX_INTEGRAL_TYPE_SIZE < size
|| integral_type_size[size] == NO_SIZE)
{
error (0, 0, _("invalid type string %s;\nthis system"
" doesn't provide a %d-byte integral type"),
quote (s_orig), size);
return false;
}
s = p;
}
break;
}

#define ISPEC_TO_FORMAT(Spec, Min_format, Long_format, Max_format) \
((Spec) == LONG_LONG ? (Max_format) \
: ((Spec) == LONG ? (Long_format) \
: (Min_format))) \
\

size_spec = integral_type_size[size];

switch (c)
{
case 'd':
fmt = SIGNED_DECIMAL;
field_width = bytes_to_signed_dec_digits[size];
sprintf (tspec->fmt_string, "%%*%s",
ISPEC_TO_FORMAT (size_spec, "d", "ld", "jd"));
break;

case 'o':
fmt = OCTAL;
sprintf (tspec->fmt_string, "%%*.%d%s",
(field_width = bytes_to_oct_digits[size]),
ISPEC_TO_FORMAT (size_spec, "o", "lo", "jo"));
break;

case 'u':
fmt = UNSIGNED_DECIMAL;
field_width = bytes_to_unsigned_dec_digits[size];

```

```

sprintf (tspec->fmt_string, "%%*%s",
        ISPEC_TO_FORMAT (size_spec, "u", "lu", "ju"));
break;

case 'x':
fmt = HEXADECIMAL;
sprintf (tspec->fmt_string, "%%*.%d%s",
        (field_width = bytes_to_hex_digits[size]),
        ISPEC_TO_FORMAT (size_spec, "x", "lx", "jx"));
break;

default:
unreachable ();
}

switch (size_spec)
{
case CHAR:
print_function = (fmt == SIGNED_DECIMAL
                  ? print_s_char
                  : print_char);
break;

case SHORT:
print_function = (fmt == SIGNED_DECIMAL
                  ? print_s_short
                  : print_short);
break;

case INT:
print_function = print_int;
break;

case LONG:
print_function = print_long;
break;

case LONG_LONG:
print_function = print_long_long;
break;

default:
affirm (false);
}
break;

case 'f':
fmt = FLOATING_POINT;
++s;
switch (*s)
{
case 'B':
++s;
fmt = BFLOATING_POINT;
size = sizeof (bfloat16);
break;

case 'H':
++s;
fmt = HFLOATING_POINT;

```

```

size = sizeof (float16);
break;

case 'F':
++s;
size = sizeof (float);
break;

case 'D':
++s;
size = sizeof (double);
break;

case 'L':
++s;
size = sizeof (long double);
break;

default:
if (! simple_strtoi (s, &p, &size))
{
/* The integer at P in S would overflow an int.
A digit string that long is sufficiently odd looking
that the following diagnostic is sufficient. */
error (0, 0, _("invalid type string %s"), quote (s_orig));
return false;
}
if (p == s)
size = sizeof (double);
else
{
if (size > MAX_FP_TYPE_SIZE
|| fp_type_size[size] == NO_SIZE
|| (! FLOAT16_SUPPORTED && BF16_SUPPORTED
&& size == sizeof (bfloat16)))
)
{
error (0, 0,
_("invalid type string %s;\n"
"this system doesn't provide a %d-byte"
" floating point type"),
quote (s_orig), size);
return false;
}
s = p;
}
break;
}
size_spec = fp_type_size[size];

if ((! FLOAT16_SUPPORTED && fmt == HFLOATING_POINT)
|| (! BF16_SUPPORTED && fmt == BFLOATING_POINT))
{
error (0, 0,
_("this system doesn't provide a %s floating point type"),
quote (s_orig));
return false;
}
{

```

```

struct lconv const *locale = localeconv ();
size_t decimal_point_len =
(locale->decimal_point[0] ? strlen (locale->decimal_point) : 1);

switch (size_spec)
{
case FLOAT_HALF:
    print_function = fmt == BFLOATING_POINT
        ? print_bfloat : print_halffloat;
    field_width = FLT_STRLEN_BOUND_L (decimal_point_len);
    break;

case FLOAT_SINGLE:
    print_function = print_float;
    field_width = FLT_STRLEN_BOUND_L (decimal_point_len);
    break;

case FLOAT_DOUBLE:
    print_function = print_double;
    field_width = DBL_STRLEN_BOUND_L (decimal_point_len);
    break;

case FLOAT_LONG_DOUBLE:
    print_function = print_long_double;
    field_width = LDBL_STRLEN_BOUND_L (decimal_point_len);
    break;

default:
    affirm (false);
}

break;
}

case 'a':
++$;
fmt = NAMED_CHARACTER;
size_spec = CHAR;
print_function = print_named_ascii;
field_width = 3;
break;

case 'c':
++$;
fmt = CHARACTER;
size_spec = CHAR;
print_function = print_ascii;
field_width = 3;
break;

default:
error (0, 0, _("invalid character '%c' in type string %s"),
      *s, quote (s_orig));
return false;
}

tspec->size = size_spec;
tspec->fmt = fmt;
tspec->print_function = print_function;

```

```

tspec->field_width = field_width;
tspec->hexl_mode_trailer = (*s == 'z');
if (tspec->hexl_mode_trailer)
    s++;

*next = s;
return true;
}

/* Given a list of one or more input filenames FILE_LIST, set the global
file pointer IN_STREAM and the global string INPUT_FILENAME to the
first one that can be successfully opened. Modify FILE_LIST to
reference the next filename in the list. A file name of "-" is
interpreted as standard input. If any file open fails, give an error
message and return false. */

static bool
open_next_file (void)
{
bool ok = true;

do
{
    input_filename = *file_list;
    if (input_filename == nullptr)
        return ok;
    ++file_list;

    if (STREQ (input_filename, "-"))
    {
        input_filename = _("standard input");
        in_stream = stdin;
        have_read_stdin = true;
        xset_binary_mode (STDIN_FILENO, O_BINARY);
    }
    else
    {
        in_stream = fopen (input_filename, (O_BINARY ? "rb" : "r"));
        if (in_stream == nullptr)
        {
            error (0, errno, "%s", quotef (input_filename));
            ok = false;
        }
    }
}
while (in_stream == nullptr);

#ifndef __MVS__
if (!isatty (fileno (in_stream)))
    __disableautocvt (fileno (in_stream));
#endif

if (limit_bytes_to_format && !flag_dump_strings)
    setvbuf (in_stream, nullptr, _IONBF, 0);

return ok;
}

/* Test whether there have been errors on in_stream, and close it if
it is not standard input. Return false if there has been an error

```

on in\_stream or stdout; return true otherwise. This function will report more than one error only if both a read and a write error have occurred. IN\_ERRNO, if nonzero, is the error number corresponding to the most recent action for IN\_STREAM. \*/

```
static bool
check_and_close (int in_errno)
{
bool ok = true;

if (in_stream != nullptr)
{
    if (!ferror (in_stream))
        in_errno = 0;
    if (STREQ (file_list[-1], "-"))
        clearerr (in_stream);
    else if (fclose (in_stream) != 0 && !in_errno)
        in_errno = errno;
    if (in_errno)
    {
        error (0, in_errno, "%s", quotef (input_filename));
        ok = false;
    }
    in_stream = nullptr;
}

if (ferror (stdout))
{
    error (0, 0, _("write error"));
    ok = false;
}

return ok;
}

/* Decode the modern od format string S. Append the decoded representation to the global array SPEC, reallocating SPEC if necessary. Return true if S is valid. */

static bool ATTRIBUTE_NONNULL ()
decode_format_string (char const *s)
{
char const *s_orig = s;

while (*s != '\0')
{
    char const *next;

    if (n_specs_allocated <= n_specs)
        spec = X2NREALLOC (spec, &n_specs_allocated);

    if (!decode_one_format (s_orig, s, &next, &spec[n_specs]))
        return false;

    affirm (s != next);
    s = next;
    ++n_specs;
}
```

```

return true;
}

/* Given a list of one or more input filenames FILE_LIST, set the global
file pointer IN_STREAM to position N_SKIP in the concatenation of
those files. If any file operation fails or if there are fewer than
N_SKIP bytes in the combined input, give an error message and return
false. When possible, use seek rather than read operations to
advance IN_STREAM. */

static bool
skip (uintmax_t n_skip)
{
bool ok = true;
int in_errno = 0;

if (n_skip == 0)
    return true;

while (in_stream != nullptr) /* EOF. */
{
    struct stat file_stats;

    /* First try seeking. For large offsets, this extra work is
worthwhile. If the offset is below some threshold it may be
more efficient to move the pointer by reading. There are two
issues when trying to seek:
- the file must be seekable.
- before seeking to the specified position, make sure
that the new position is in the current file.
Try to do that by getting file's size using fstat.
But that will work only for regular files. */

    if (fstat (fileno (in_stream), &file_stats) == 0)
    {
        bool usable_size = usable_st_size (&file_stats);

        /* The st_size field is valid for regular files.
If the number of bytes left to skip is larger than
the size of the current file, we can decrement n_skip
and go on to the next file. Skip this optimization also
when st_size is no greater than the block size, because
some kernels report nonsense small file sizes for
proc-like file systems. */
        if (usable_size && STP_BLKSIZ (&file_stats) < file_stats.st_size)
        {
            if ((uintmax_t) file_stats.st_size < n_skip)
                n_skip -= file_stats.st_size;
            else
            {
                if (fseeko (in_stream, n_skip, SEEK_CUR) != 0)
                {
                    in_errno = errno;
                    ok = false;
                }
                n_skip = 0;
            }
        }
    }
}

else if (!usable_size && fseeko (in_stream, n_skip, SEEK_CUR) == 0)

```

```

n_skip = 0;

/* If it's not a regular file with nonnegative size,
   or if it's so small that it might be in a proc-like file system,
   position the file pointer by reading. */

else
{
    char buf[BUFSIZ];
    size_t n_bytes_read, n_bytes_to_read = BUFSIZ;

    while (0 < n_skip)
    {
        if (n_skip < n_bytes_to_read)
            n_bytes_to_read = n_skip;
        n_bytes_read = fread (buf, 1, n_bytes_to_read, in_stream);
        n_skip -= n_bytes_read;
        if (n_bytes_read != n_bytes_to_read)
        {
            if (ferror (in_stream))
            {
                in_errno = errno;
                ok = false;
                n_skip = 0;
                break;
            }
            if (feof (in_stream))
                break;
        }
    }

    if (n_skip == 0)
        break;
}

else /* cannot fstat() file */
{
    error (0, errno, "%s", quotef (input_filename));
    ok = false;
}

ok &= check_and_close (in_errno);

ok &= open_next_file ();
}

if (n_skip != 0)
    error (EXIT_FAILURE, 0, _("cannot skip past end of combined input"));

return ok;
}

static void
format_address_none (MAYBE_UNUSED uintmax_t address,
                     MAYBE_UNUSED char c)
{
}

static void

```

```

format_address_std (uintmax_t address, char c)
{
    char buf[MAX_ADDRESS_LENGTH + 2];
    char *p = buf + sizeof buf;
    char const *pbound;

    *--p = '\0';
    *--p = c;
    pbound = p - address_pad_len;

    /* Use a special case of the code for each base. This is measurably
       faster than generic code. */
    switch (address_base)
    {
        case 8:
        do
            *--p = '0' + (address & 7);
        while ((address >= 3) != 0);
        break;

        case 10:
        do
            *--p = '0' + (address % 10);
        while ((address /= 10) != 0);
        break;

        case 16:
        do
            *--p = "0123456789abcdef"[address & 15];
        while ((address >= 4) != 0);
        break;
    }

    while (pbound < p)
        *--p = '0';

    fputs (p, stdout);
}

static void
format_address_paren (uintmax_t address, char c)
{
    putchar ('(');
    format_address_std (address, ')');
    if (c)
        putchar (c);
}

static void
format_address_label (uintmax_t address, char c)
{
    format_address_std (address, ' ');
    format_address_paren (address + pseudo_offset, c);
}

/* Write N_BYTES bytes from CURR_BLOCK to standard output once for each
   of the N_SPEC format specs. CURRENT_OFFSET is the byte address of
   CURR_BLOCK in the concatenation of input files, and it is printed
   (optionally) only before the output line associated with the first
   format spec. When duplicate blocks are being abbreviated, the output

```

for a sequence of identical input blocks is the output for the first block followed by an asterisk alone on a line. It is valid to compare the blocks PREV\_BLOCK and CURR\_BLOCK only when N\_BYTES == BYTES\_PER\_BLOCK.

That condition may be false only for the last input block. \*/

```

static void
write_block (uintmax_t current_offset, size_t n_bytes,
            char const *prev_block, char const *curr_block)
{
    static bool first = true;
    static bool prev_pair_equal = false;

#define EQUAL_BLOCKS(b1, b2) (memcmp (b1, b2, bytes_per_block) == 0)

if (abbreviate_duplicate_blocks
    && !first && n_bytes == bytes_per_block
    && EQUAL_BLOCKS (prev_block, curr_block))
{
    if (prev_pair_equal)
    {
        /* The two preceding blocks were equal, and the current
         * block is the same as the last one, so print nothing. */
    }
    else
    {
        printf ("*\n");
        prev_pair_equal = true;
    }
}
else
{
    prev_pair_equal = false;
    for (size_t i = 0; i < n_specs; i++)
    {
        int datum_width = width_bytes[spec[i].size];
        int fields_per_block = bytes_per_block / datum_width;
        int blank_fields = (bytes_per_block - n_bytes) / datum_width;
        if (i == 0)
            format_address (current_offset, '\0');
        else
            printf ("%*s", address_pad_len, "");
        (*spec[i].print_function) (fields_per_block, blank_fields,
                                  curr_block, spec[i].fmt_string,
                                  spec[i].field_width, spec[i].pad_width);
        if (spec[i].hexl_mode_trailer)
        {
            /* space-pad out to full line width, then dump the trailer */
            int field_width = spec[i].field_width;
            int pad_width = (spec[i].pad_width * blank_fields
                            / fields_per_block);
            printf ("%*s", blank_fields * field_width + pad_width, "");
            dump_hexl_mode_trailer (n_bytes, curr_block);
        }
        putchar ('\n');
    }
}
first = false;
}

```

```

/* Read a single byte into *C from the concatenation of the input files
named in the global array FILE_LIST. On the first call to this
function, the global variable IN_STREAM is expected to be an open
stream associated with the input file INPUT_FILENAME. If IN_STREAM
is at end-of-file, close it and update the global variables IN_STREAM
and INPUT_FILENAME so they correspond to the next file in the list.
Then try to read a byte from the newly opened file. Repeat if
necessary until EOF is reached for the last file in FILE_LIST, then
set *C to EOF and return. Subsequent calls do likewise. Return
true if successful. */

static bool
read_char (int *c)
{
    bool ok = true;

    *c = EOF;

    while (in_stream != nullptr) /* EOF. */
    {
        *c = fgetc (in_stream);

        if (*c != EOF)
            break;

        ok &= check_and_close (errno);

        ok &= open_next_file ();
    }

    return ok;
}

/* Read N bytes into BLOCK from the concatenation of the input files
named in the global array FILE_LIST. On the first call to this
function, the global variable IN_STREAM is expected to be an open
stream associated with the input file INPUT_FILENAME. If all N
bytes cannot be read from IN_STREAM, close IN_STREAM and update
the global variables IN_STREAM and INPUT_FILENAME. Then try to
read the remaining bytes from the newly opened file. Repeat if
necessary until EOF is reached for the last file in FILE_LIST.
On subsequent calls, don't modify BLOCK and return true. Set
*N_BYTGES_IN_BUFFER to the number of bytes read. If an error occurs,
it will be detected through ferror when the stream is about to be
closed. If there is an error, give a message but continue reading
as usual and return false. Otherwise return true. */

static bool
read_block (size_t n, char *block, size_t *n_bytes_in_buffer)
{
    bool ok = true;

    affirm (0 < n && n <= bytes_per_block);

    *n_bytes_in_buffer = 0;

    while (in_stream != nullptr) /* EOF. */
    {
        size_t n_needed;
        size_t n_read;

```

```

n_needed = n - *n_bytes_in_buffer;
n_read = fread (block + *n_bytes_in_buffer, 1, n_needed, in_stream);

*n_bytes_in_buffer += n_read;

if (n_read == n_needed)
    break;

ok &= check_and_close (errno);

ok &= open_next_file ();
}

return ok;
}

/* Return the least common multiple of the sizes associated
with the format specs. */

ATTRIBUTE_PURE
static int
get_lcm (void)
{
int l_c_m = 1;

for (size_t i = 0; i < n_specs; i++)
    l_c_m = lcm (l_c_m, width_bytes[spec[i].size]);
return l_c_m;
}

/* If S is a valid traditional offset specification with an optional
leading '+' return true and set *OFFSET to the offset it denotes. */

static bool
parse_old_offset (char const *s, uintmax_t *offset)
{
int radix;

if (*s == '\0')
    return false;

/* Skip over any leading '+'. */
if (s[0] == '+')
    ++s;

/* Determine the radix we'll use to interpret S. If there is a '.', 
it's decimal, otherwise, if the string begins with '0X'or '0x',
it's hexadecimal, else octal. */
if (strchr (s, '.') != nullptr)
    radix = 10;
else
{
    if (s[0] == '0' && (s[1] == 'x' || s[1] == 'X'))
        radix = 16;
    else
        radix = 8;
}

return xstrtoumax (s, nullptr, radix, offset, "Bb") == LONGINT_OK;
}

```

```

}

/* Read a chunk of size BYTES_PER_BLOCK from the input files, write the
formatted block to standard output, and repeat until the specified
maximum number of bytes has been read or until all input has been
processed. If the last block read is smaller than BYTES_PER_BLOCK
and its size is not a multiple of the size associated with a format
spec, extend the input block with zero bytes until its length is a
multiple of all format spec sizes. Write the final block. Finally,
write on a line by itself the offset of the byte after the last byte
read. Accumulate return values from calls to read_block and
check_and_close, and if any was false, return false.
Otherwise, return true. */

static bool
dump (void)
{
char *block[2];
uintmax_t current_offset;
bool idx = false;
bool ok = true;
size_t n_bytes_read;

block[0] = xnmalloc (2, bytes_per_block);
block[1] = block[0] + bytes_per_block;

current_offset = n_bytes_to_skip;

if (limit_bytes_to_format)
{
    while (ok)
    {
        size_t n_needed;
        if (current_offset >= end_offset)
        {
            n_bytes_read = 0;
            break;
        }
        n_needed = MIN (end_offset - current_offset,
                        (uintmax_t) bytes_per_block);
        ok &= read_block (n_needed, block[idx], &n_bytes_read);
        if (n_bytes_read < bytes_per_block)
            break;
        affirm (n_bytes_read == bytes_per_block);
        write_block (current_offset, n_bytes_read,
                    block[!idx], block[idx]);
        if (ferror (stdout))
            ok = false;
        current_offset += n_bytes_read;
        idx = !idx;
    }
}
else
{
    while (ok)
    {
        ok &= read_block (bytes_per_block, block[idx], &n_bytes_read);
        if (n_bytes_read < bytes_per_block)
            break;
        affirm (n_bytes_read == bytes_per_block);
    }
}

```

```

        write_block (current_offset, n_bytes_read,
                     block[!idx], block[idx]);
        if (ferror (stdout))
            ok = false;
        current_offset += n_bytes_read;
        idx = !idx;
    }
}

if (n_bytes_read > 0)
{
    int l_c_m;
    size_t bytes_to_write;

    l_c_m = get_lcm ();

    /* Ensure zero-byte padding up to the smallest multiple of l_c_m that
       is at least as large as n_bytes_read. */
    bytes_to_write = l_c_m * ((n_bytes_read + l_c_m - 1) / l_c_m);

    memset (block[idx] + n_bytes_read, 0, bytes_to_write - n_bytes_read);
    write_block (current_offset, n_bytes_read, block[!idx], block[idx]);
    current_offset += n_bytes_read;
}

format_address (current_offset, '\n');

if (limit_bytes_to_format && current_offset >= end_offset)
    ok &= check_and_close (0);

free (block[0]);

return ok;
}

/* STRINGS mode. Find each "string constant" in the input.
A string constant is a run of at least 'string_min' ASCII
graphic (or formatting) characters terminated by a null.
Based on a function written by Richard Stallman for a
traditional version of od. Return true if successful. */

static bool
dump_strings (void)
{
size_t bufsize = MAX (100, string_min);
char *buf = xmalloc (bufsize);
uintmax_t address = n_bytes_to_skip;
bool ok = true;

while (true)
{
    size_t i;
    int c;

    /* See if the next 'string_min' chars are all printing chars. */
    tryline:

    if (limit_bytes_to_format
        && (end_offset < string_min || end_offset - string_min <= address))
        break;
}

```

```

for (i = 0; i < string_min; i++)
{
    ok &= read_char (&c);
    address++;
    if (c < 0)
    {
        free (buf);
        return ok;
    }
    if (!isprint (c))
        /* Found a non-printing. Try again starting with next char. */
        goto tryline;
    buf[i] = c;
}

/* We found a run of 'string_min' printable characters.
   Now see if it is terminated with a null byte. */
while (!limit_bytes_to_format || address < end_offset)
{
    if (i == bufsize)
    {
        buf = X2REALLOC (buf, &bufsize);
    }
    ok &= read_char (&c);
    address++;
    if (c < 0)
    {
        free (buf);
        return ok;
    }
    if (c == '\0')
        break;          /* It is; print this string. */
    if (!isprint (c))
        goto tryline; /* It isn't; give up on this string. */
    buf[i++] = c;           /* String continues; store it all. */
}

/* If we get here, the string is all printable and null-terminated,
   so print it. It is all in 'buf' and 'i' is its length. */
buf[i] = 0;
format_address (address - i - 1, ' ');

for (i = 0; (c = buf[i]); i++)
{
    switch (c)
    {
        case '\a':
            fputs ("\\\a", stdout);
            break;

        case '\b':
            fputs ("\\\b", stdout);
            break;

        case '\f':
            fputs ("\\\f", stdout);
            break;

        case '\n':

```

```

fputs ("\n", stdout);
break;

case '\r':
fputs ("\r", stdout);
break;

case '\t':
fputs ("\t", stdout);
break;

case '\v':
fputs ("\v", stdout);
break;

default:
putc (c, stdout);
}
}

putchar ('\n');
}

/* We reach this point only if we search through
(max_bytes_to_format - string_min) bytes before reaching EOF. */

free (buf);

ok &= check_and_close (0);
return ok;
}

int
main (int argc, char **argv)
{
int n_files;
size_t i;
int l_c_m;
idx_t desired_width IF_LINT (= 0);
bool modern = false;
bool width_specified = false;
bool ok = true;
size_t width_per_block = 0;
static char const multipliers[] = "bEGKkMmPQRTYZ0";

/* The old-style 'pseudo starting address' to be printed in parentheses
after any true address. */
uintmax_t pseudo_start IF_LINT (= 0);

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

atexit (close_stdout);

for (i = 0; i <= MAX_INTEGRAL_TYPE_SIZE; i++)
    integral_type_size[i] = NO_SIZE;

integral_type_size[sizeof (char)] = CHAR;

```



```

address_base = 16;
address_pad_len = 6;
break;
case 'n':
format_address = format_address_none;
address_pad_len = 0;
break;
default:
error (EXIT_FAILURE, 0,
      _("invalid output address radix '%c';"
        " it must be one character from [doxn]"),
      optarg[0]);
break;
}
break;

case 'j':
modern = true;
s_err = xstrtoumax (optarg, nullptr, 0,
                     &n_bytes_to_skip, multipliers);
if (s_err != LONGINT_OK)
    xstrtol_fatal (s_err, oi, c, long_options, optarg);
break;

case 'N':
modern = true;
limit_bytes_to_format = true;

s_err = xstrtoumax (optarg, nullptr, 0, &max_bytes_to_format,
                     multipliers);
if (s_err != LONGINT_OK)
    xstrtol_fatal (s_err, oi, c, long_options, optarg);
break;

case 'S':
modern = true;
if (optarg == nullptr)
    string_min = 3;
else
{
    s_err = xstrtoumax (optarg, nullptr, 0, &tmp, multipliers);
    if (s_err != LONGINT_OK)
        xstrtol_fatal (s_err, oi, c, long_options, optarg);

/* The minimum string length may be no larger than SIZE_MAX,
   since we may allocate a buffer of this size. */
    if (SIZE_MAX < tmp)
        error (EXIT_FAILURE, 0, _("'%s is too large"), quote (optarg));

    string_min = tmp;
}
flag_dump_strings = true;
break;

case 't':
modern = true;
ok &= decode_format_string (optarg);
break;

case 'v':

```

```

modern = true;
abbreviate_duplicate_blocks = false;
break;

case TRADITIONAL_OPTION:
traditional = true;
break;

case ENDIAN_OPTION:
switch (XARGMATCH ("--endian", optarg, endian_args, endian_types))
{
  case endian_big:
    input_swap = ! WORDS_BIGENDIAN;
    break;
  case endian_little:
    input_swap = WORDS_BIGENDIAN;
    break;
}
break;

/* The next several cases map the traditional format
   specification options to the corresponding modern format
   specs.  GNU od accepts any combination of old- and
   new-style options.  Format specification options accumulate.
   The obsolescent and undocumented formats are compatible
   with FreeBSD 4.10 od. */

#define CASE_OLD_ARG(old_char,new_string) \
  case old_char: \
    ok &= decode_format_string (new_string); \
    break \
  \
CASE_OLD_ARG ('a', "a"); \
CASE_OLD_ARG ('b', "o1"); \
CASE_OLD_ARG ('c', "c"); \
CASE_OLD_ARG ('D', "u4"); /* obsolescent and undocumented */ \
CASE_OLD_ARG ('d', "u2"); \
case 'F': /* obsolescent and undocumented alias */ \
CASE_OLD_ARG ('e', "fD"); /* obsolescent and undocumented */ \
CASE_OLD_ARG ('f', "fF"); \
case 'X': /* obsolescent and undocumented alias */ \
CASE_OLD_ARG ('H', "x4"); /* obsolescent and undocumented */ \
CASE_OLD_ARG ('i', "di"); \
case 'I': case 'L': /* obsolescent and undocumented aliases */ \
CASE_OLD_ARG ('I', "dL"); \
CASE_OLD_ARG ('O', "o4"); /* obsolescent and undocumented */ \
case 'B': /* obsolescent and undocumented alias */ \
CASE_OLD_ARG ('o', "o2"); \
CASE_OLD_ARG ('s', "d2"); \
case 'h': /* obsolescent and undocumented alias */ \
CASE_OLD_ARG ('x', "x2");

#undef CASE_OLD_ARG

case 'w':
modern = true;
width_specified = true;
if (optarg == nullptr)
{
  desired_width = 32;
}

```

```

        }
    else
    {
        intmax_t w_tmp;
        s_err = xstrtoimax (optarg, nullptr, 10, &w_tmp, "");
        if (s_err != LONGINT_OK || w_tmp <= 0)
            xstrtol_fatal (s_err, oi, c, long_options, optarg);
        if (ckd_add (&desired_width, w_tmp, 0))
            error (EXIT_FAILURE, 0, _("'%s is too large"), quote (optarg));
    }
    break;

    case getopt_HELP_CHAR;

    case getopt_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

    default:
        usage (EXIT_FAILURE);
        break;
    }
}

if (!ok)
    return EXIT_FAILURE;

if (flag_dump_strings && n_specs > 0)
    error (EXIT_FAILURE, 0,
           _("no type may be specified when dumping strings"));

n_files = argc - optind;

/* If the --traditional option is used, there may be from
   0 to 3 remaining command line arguments; handle each case
   separately.
   od [file] [[+]offset[.]][b] [[+]label[.]][b]]
The offset and label have the same syntax.

If --traditional is not given, and if no modern options are
given, and if the offset begins with + or (if there are two
operands) a digit, accept only this form, as per POSIX:
   od [file] [[+]offset[.]][b]]
*/
if (!modern || traditional)
{
    uintmax_t o1;
    uintmax_t o2;

    switch (n_files)
    {
        case 1:
            if ((traditional || argv[optind][0] == '+')
                && parse_old_offset (argv[optind], &o1))
            {
                n_bytes_to_skip = o1;
                --n_files;
                ++argv;
            }
    }
    break;
}

```

```

case 2:
if ((traditional || argv[optind + 1][0] == '+'
    || ISDIGIT (argv[optind + 1][0]))
    && parse_old_offset (argv[optind + 1], &o2))
{
    if (traditional && parse_old_offset (argv[optind], &o1))
    {
        n_bytes_to_skip = o1;
        flag_pseudo_start = true;
        pseudo_start = o2;
        argv += 2;
        n_files -= 2;
    }
    else
    {
        n_bytes_to_skip = o2;
        --n_files;
        argv[optind + 1] = argv[optind];
        ++argv;
    }
}
break;

case 3:
if (traditional
    && parse_old_offset (argv[optind + 1], &o1)
    && parse_old_offset (argv[optind + 2], &o2))
{
    n_bytes_to_skip = o1;
    flag_pseudo_start = true;
    pseudo_start = o2;
    argv[optind + 2] = argv[optind];
    argv += 2;
    n_files -= 2;
}
break;
}

if (traditional && 1 < n_files)
{
    error (0, 0, _("extra operand %s"), quote (argv[optind + 1]));
    error (0, 0, "%s",
           _("compatibility mode supports at most one file"));
    usage (EXIT_FAILURE);
}

if (flag_pseudo_start)
{
    if (format_address == format_address_none)
    {
        address_base = 8;
        address_pad_len = 7;
        format_address = format_address_paren;
    }
    else
        format_address = format_address_label;
}

if (limit_bytes_to_format)

```

```

{
end_offset = n_bytes_to_skip + max_bytes_to_format;
if (end_offset < n_bytes_to_skip)
    error (EXIT_FAILURE, 0, _("skip-bytes + read-bytes is too large"));
}

if (n_specs == 0)
    decode_format_string ("oS");

if (n_files > 0)
{
/* Set the global pointer FILE_LIST so that it
   references the first file-argument on the command-line. */

file_list = (char const *const *)&argv[optind];
}
else
{
/* No files were listed on the command line.
   Set the global pointer FILE_LIST so that it
   references the null-terminated list of one name: "-". */

file_list = default_file_list;
}

/* open the first input file */
ok = open_next_file ();
if (in_stream == nullptr)
    goto cleanup;

/* skip over any unwanted header bytes */
ok &= skip (n_bytes_to_skip);
if (in_stream == nullptr)
    goto cleanup;

pseudo_offset = (flag_pseudo_start ? pseudo_start - n_bytes_to_skip : 0);

/* Compute output block length. */
l_c_m = get_lcm ();

if (width_specified)
{
    if (desired_width != 0 && desired_width % l_c_m == 0)
        bytes_per_block = desired_width;
    else
    {
        {
            error (0, 0, _("warning: invalid width %td; using %d instead"),
                  desired_width, l_c_m);
            bytes_per_block = l_c_m;
        }
    }
}
else
{
    if (l_c_m < DEFAULT_BYTES_PER_BLOCK)
        bytes_per_block = l_c_m * (DEFAULT_BYTES_PER_BLOCK / l_c_m);
    else
        bytes_per_block = l_c_m;
}

/* Compute padding necessary to align output block. */

```

```

for (i = 0; i < n_specs; i++)
{
    int fields_per_block = bytes_per_block / width_bytes[spec[i].size];
    int block_width = (spec[i].field_width + 1) * fields_per_block;
    if (width_per_block < block_width)
        width_per_block = block_width;
}
for (i = 0; i < n_specs; i++)
{
    int fields_per_block = bytes_per_block / width_bytes[spec[i].size];
    int block_width = spec[i].field_width * fields_per_block;
    spec[i].pad_width = width_per_block - block_width;
}

#ifndef DEBUG
printf ("lcm=%d, width_per_block=%zu\n", l_c_m, width_per_block);
for (i = 0; i < n_specs; i++)
{
    int fields_per_block = bytes_per_block / width_bytes[spec[i].size];
    affirm (bytes_per_block % width_bytes[spec[i].size] == 0);
    affirm (1 <= spec[i].pad_width / fields_per_block);
    printf ("%d: fmt=\"%s\" in_width=%d out_width=%d pad=%d\n",
           i, spec[i].fmt_string, width_bytes[spec[i].size],
           spec[i].field_width, spec[i].pad_width);
}
#endif

ok &= (flag_dump_strings ? dump_strings () : dump ());

cleanup:

if (have_read_stdin && fclose (stdin) == EOF)
    error (EXIT_FAILURE, errno, _("standard input"));

return ok ? EXIT_SUCCESS : EXIT_FAILURE;
"""

,
"patch":
"""
diff --git a/src/od.c b/src/od.c
index d23df2c..986b6a5 100644
--- a/src/od.c
+++ b/src/od.c
@@ -981,6 +981,11 @@ open_next_file (void)
}
while (in_stream == nullptr);

+__ifdef __MVS__
+ if (!isatty(fileno(in_stream)))
+     __disableautocvt(fileno(in_stream));
+__endif
+
+ if (limit_bytes_to_format && !flag_dump_strings)
+     setvbuf (in_stream, nullptr, _IONBF, 0);
"""

},
{
    "wrong_code": "
"""

```

```
/* timeout -- run a command with bounded time
Copyright (C) 2008-2024 Free Software Foundation, Inc.
```

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>. \*/

```
/* timeout - Start a command, and kill it if the specified timeout expires
```

We try to behave like a shell starting a single (foreground) job, and will kill the job if we receive the alarm signal we setup.

The exit status of the job is returned, or one of these errors:

```
EXIT_TIMEDOUT 124    job timed out
EXIT_CANCELED 125    internal error
EXIT_CANNOT_INVOKE 126  error executing job
EXIT_ENOENT   127    couldn't find job to exec
```

Caveats:

If user specifies the KILL (9) signal is to be sent on timeout, the monitor is killed and so exits with 128+9 rather than 124.

If you start a command in the background, which reads from the tty and so is immediately sent SIGTTIN to stop, then the timeout process will ignore this so it can timeout the command as expected. This can be seen with 'timeout 10 dd&' for example.

However if one brings this group to the foreground with the 'fg' command before the timer expires, the command will remain in the stop state as the shell doesn't send a SIGCONT because the timeout process (group leader) is already running.

To get the command running again one can Ctrl-Z, and do fg again. Note one can Ctrl-C the whole job when in this state.

I think this could be fixed but I'm not sure the extra complication is justified for this scenario.

Written by Pádraig Brady. \*/

```
#include <config.h>
#include <getopt.h>
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#if HAVE_PRCTL
# include <sys/prctl.h>
#endif
#include <sys/wait.h>

#include "system.h"
#include "cl-strtod.h"
#include "xstrtod.h"
#include "sig2str.h"
```

```

#include "operand2sig.h"
#include "quote.h"

#if HAVE_SETRLIMIT
/* FreeBSD 5.0 at least needs <sys/types.h> and <sys/time.h> included
before <sys/resource.h>. Currently "system.h" includes <sys/time.h>. */
#include <sys/resource.h>
#endif

/* NonStop circa 2011 lacks both SA_RESTART and siginterrupt. */
#ifndef SA_RESTART
#define SA_RESTART 0
#endif

#define PROGRAM_NAME "timeout"

#define AUTHORS proper_name_lite ("Padraig Brady", "P\303\241draig Brady")

static int timed_out;
static int term_signal = SIGTERM; /* same default as kill command. */
static pid_t monitored_pid;
static double kill_after;
static bool foreground; /* whether to use another program group. */
static bool preserve_status; /* whether to use a timeout status or not. */
static bool verbose; /* whether to diagnose timeouts or not. */
static char const *command;

/* for long options with no corresponding short option, use enum */
enum
{
    FOREGROUND_OPTION = CHAR_MAX + 1,
    PRESERVE_STATUS_OPTION
};

static struct option const long_options[] =
{
    {"kill-after", required_argument, nullptr, 'k'},
    {"signal", required_argument, nullptr, 's'},
    {"verbose", no_argument, nullptr, 'v'},
    {"foreground", no_argument, nullptr, FOREGROUND_OPTION},
    {"preserve-status", no_argument, nullptr, PRESERVE_STATUS_OPTION},
    {GETOPT_HELP_OPTION_DECL},
    {GETOPT_VERSION_OPTION_DECL},
    {nullptr, 0, nullptr, 0}
};

/* Start the timeout after which we'll receive a SIGALRM.
Round DURATION up to the next representable value.
Treat out-of-range values as if they were maximal,
as that's more useful in practice than reporting an error.
'0' means don't timeout. */
static void
settimeout (double duration, bool warn)
{

#if HAVE_TIMER_SETTIME
/* timer_settime() provides potentially nanosecond resolution. */

struct timespec ts = dtotimespec (duration);
struct itimerspec its = {.it_interval = {0}, .it_value = ts};

```

```

timer_t timerid;
if (timer_create (CLOCK_REALTIME, nullptr, &timerid) == 0)
{
    if (timer_settime (timerid, 0, &its, nullptr) == 0)
        return;
    else
    {
        if (warn)
            error (0, errno, _("warning: timer_settime"));
        timer_delete (timerid);
    }
}
else if (warn && errno != ENOSYS)
    error (0, errno, _("warning: timer_create"));

#elif HAVE_SETITIMER
/* setitimer() is more portable (to Darwin for example),
   but only provides microsecond resolution. */

struct timeval tv;
struct timespec ts = dtotimespec (duration);
tv.tv_sec = ts.tv_sec;
tv.tv_usec = (ts.tv_nsec + 999) / 1000;
if (tv.tv_usec == 1000 * 1000)
{
    if (tv.tv_sec != TYPE_MAXIMUM (time_t))
    {
        tv.tv_sec++;
        tv.tv_usec = 0;
    }
    else
        tv.tv_usec--;
}
struct itimerval it = { .it_interval = {0}, .it_value = tv };
if (setitimer (ITIMER_REAL, &it, nullptr) == 0)
    return;
else
{
    if (warn && errno != ENOSYS)
        error (0, errno, _("warning: setitimer"));
}
#endif

/* fallback to single second resolution provided by alarm(). */

unsigned int timeint;
if (UINT_MAX <= duration)
    timeint = UINT_MAX;
else
{
    unsigned int duration_floor = duration;
    timeint = duration_floor + (duration_floor < duration);
}
alarm (timeint);
}

/* send SIG avoiding the current process. */

static int
send_sig (pid_t where, int sig)

```

```

{
/* If sending to the group, then ignore the signal,
so we don't go into a signal loop. Note that this will ignore any of the
signals registered in install_cleanup(), that are sent after we
propagate the first one, which hopefully won't be an issue. Note this
process can be implicitly multithreaded due to some timer_settime()
implementations, therefore a signal sent to the group, can be sent
multiple times to this process. */
if (where == 0)
    signal (sig, SIG_IGN);
return kill (where, sig);
}

/* Signal handler which is required for sigsuspend() to be interrupted
whenever SIGCHLD is received. */
static void
chld (int sig)
{
}

static void
cleanup (int sig)
{
if (sig == SIGALRM)
{
    timed_out = 1;
    sig = term_signal;
}
if (0 < monitored_pid)
{
    if (kill_after)
    {
        int saved_errno = errno; /* settimeout may reset. */
        /* Start a new timeout after which we'll send SIGKILL. */
        term_signal = SIGKILL;
        settimeout (kill_after, false);
        kill_after = 0; /* Don't let later signals reset kill alarm. */
        errno = saved_errno;
    }
}

/* Send the signal directly to the monitored child,
in case it has itself become group leader,
or is not running in a separate group. */
if (verbose)
{
    char signame[MAX (SIG2STR_MAX, INT_BUFSIZE_BOUND (int))];
    if (sig2str (sig, signame) != 0)
        snprintf (signame, sizeof signame, "%d", sig);
    error (0, 0, _("sending signal %s to command %s"),
           signame, quote (command));
}
send_sig (monitored_pid, sig);

/* The normal case is the job has remained in our
newly created process group, so send to all processes in that. */
if (!foreground)
{
    send_sig (0, sig);
    if (sig != SIGKILL && sig != SIGCONT)

```

```

        {
            send_sig (monitored_pid, SIGCONT);
            send_sig (0, SIGCONT);
        }
    }

else if (monitored_pid == -1)
    /* were in the parent, so let it continue to exit below. */
}
else /* monitored_pid == 0 */
    /* parent hasn't forked yet, or child has not exec'd yet. */
    _exit (128 + sig);
}

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    printf (_("\
Usage: %s [OPTION] DURATION COMMAND [ARG]...\n\
or: %s [OPTION]\n"), program_name, program_name);

    fputs (_("\
Start COMMAND, and kill it if still running after DURATION.\n\
"), stdout);

    emit_mandatory_arg_note ();

    fputs (_("\
--preserve-status\n\
      exit with the same status as COMMAND, even when the\n\
      command times out\n\
--foreground\n\
      when not running timeout directly from a shell prompt,\n\
      allow COMMAND to read from the TTY and get TTY signals;\n\
      in this mode, children of COMMAND will not be timed out\n\
-k, --kill-after=DURATION\n\
      also send a KILL signal if COMMAND is still running\n\
      this long after the initial signal was sent\n\
-s, --signal=SIGNAL\n\
      specify the signal to be sent on timeout;\n\
      SIGNAL may be a name like 'HUP' or a number;\n\
      see 'kill -l' for a list of signals\n"), stdout);

    fputs (_("\
-v, --verbose  diagnose to stderr any signal sent upon timeout\n"), stdout);

    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);

    fputs (_("\n\
DURATION is a floating point number with an optional suffix:\n\
's' for seconds (the default), 'm' for minutes, 'h' for hours or \
'd' for days.\nA duration of 0 disables the associated timeout.\n"), stdout);

    fputs (_("\n\
Upon timeout, send the TERM signal to COMMAND, if no other SIGNAL specified.\n\

```

The TERM signal kills any process that does not block or catch that signal.\n\\ It may be necessary to use the KILL signal, since this signal can't be caught.\n\\ \n"), stdout);

```
    fputs (_("\n\\
Exit status:\n\\
124 if COMMAND times out, and --preserve-status is not specified\n\\
125 if the timeout command itself fails\n\\
126 if COMMAND is found but cannot be invoked\n\\
127 if COMMAND cannot be found\n\\
137 if COMMAND (or timeout itself) is sent the KILL (9) signal (128+9)\n\\
- the exit status of COMMAND otherwise\n\\
"), stdout);
```

```
    emit_ancillary_info (PROGRAM_NAME);
}
exit (status);
}
```

```
/* Given a floating point value *X, and a suffix character, SUFFIX_CHAR,
scale *X by the multiplier implied by SUFFIX_CHAR. SUFFIX_CHAR may
be the NUL byte or 's' to denote seconds, 'm' for minutes, 'h' for
hours, or 'd' for days. If SUFFIX_CHAR is invalid, don't modify *X
and return false. Otherwise return true. */
```

```
static bool
apply_time_suffix (double *x, char suffix_char)
{
int multiplier;

switch (suffix_char)
{
case 0:
case 's':
multiplier = 1;
break;
case 'm':
multiplier = 60;
break;
case 'h':
multiplier = 60 * 60;
break;
case 'd':
multiplier = 60 * 60 * 24;
break;
default:
return false;
}
```

```
*x *= multiplier;
```

```
return true;
}
```

```
static double
parse_duration (char const *str)
{
double duration;
char const *ep;
```

```

if (! (xstrtod (str, &ep, &duration, cl strtod) || errno == ERANGE)
    /* Nonnegative interval. */
    || !(0 <= duration)
    /* No extra chars after the number and an optional s,m,h,d char. */
    || (*ep && *(ep + 1))
    /* Check any suffix char and update timeout based on the suffix. */
    || !apply_time_suffix (&duration, *ep))
{
    error (0, 0, _("invalid time interval %s")), quote (str);
    usage (EXIT_CANCELED);
}

return duration;
}

static void
unblock_signal (int sig)
{
sigset_t unblock_set;
sigemptyset (&unblock_set);
sigaddset (&unblock_set, sig);
if (sigprocmask (SIG_UNBLOCK, &unblock_set, nullptr) != 0)
    error (0, errno, _("warning: sigprocmask"));
}

static void
install_sigchld (void)
{
struct sigaction sa;
sigemptyset (&sa.sa_mask); /* Allow concurrent calls to handler */
sa.sa_handler = chld;
sa.sa_flags = SA_RESTART; /* Restart syscalls if possible, as that's
                           more likely to work cleanly. */

sigaction (SIGCHLD, &sa, nullptr);

/* We inherit the signal mask from our parent process,
   so ensure SIGCHLD is not blocked. */
unblock_signal (SIGCHLD);
}

static void
install_cleanup (int sigterm)
{
struct sigaction sa;
sigemptyset (&sa.sa_mask); /* Allow concurrent calls to handler */
sa.sa_handler = cleanup;
sa.sa_flags = SA_RESTART; /* Restart syscalls if possible, as that's
                           more likely to work cleanly. */

sigaction (SIGALRM, &sa, nullptr); /* our timeout. */
sigaction (SIGINT, &sa, nullptr); /* Ctrl-C at terminal for example. */
sigaction (SIGQUIT, &sa, nullptr); /* Ctrl-\ at terminal for example. */
sigaction (SIGHUP, &sa, nullptr); /* terminal closed for example. */
sigaction (SIGTERM, &sa, nullptr); /* if killed, stop monitored proc. */
sigaction (sigterm, &sa, nullptr); /* user specified termination signal. */
}

/* Block all signals which were registered with cleanup() as the signal
   handler, so we never kill processes after waitpid() returns.
}

```

```

Also block SIGCHLD to ensure it doesn't fire between
waitpid() polling and sigsuspend() waiting for a signal.
Return original mask in OLD_SET. */
static void
block_cleanup_and_chld (int sigterm, sigset_t *old_set)
{
sigset_t block_set;
sigemptyset (&block_set);

sigaddset (&block_set, SIGALRM);
sigaddset (&block_set, SIGINT);
sigaddset (&block_set, SIGQUIT);
sigaddset (&block_set, SIGHUP);
sigaddset (&block_set, SIGTERM);
sigaddset (&block_set, sigterm);

sigaddset (&block_set, SIGCHLD);

if (sigprocmask (SIG_BLOCK, &block_set, old_set) != 0)
    error (0, errno, _("warning: sigprocmask"));
}

/* Try to disable core dumps for this process.
Return TRUE if successful, FALSE otherwise. */
static bool
disable_core_dumps (void)
{
#if HAVE_PRCTL && defined PR_SET_DUMPABLE
if (prctl (PR_SET_DUMPABLE, 0) == 0)
    return true;

#elif HAVE_SETRLIMIT && defined RLIMIT_CORE
/* Note this doesn't disable processing by a filter in
   /proc/sys/kernel/core_pattern on Linux. */
if (setrlimit (RLIMIT_CORE, &(struct rlimit) {0}) == 0)
    return true;
#endif

#else
return false;
#endif

error (0, errno, _("warning: disabling core dumps failed"));
return false;
}

int
main (int argc, char **argv)
{
double timeout;
int c;

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

initialize_exit_failure (EXIT_CANCELED);
atexit (close_stdout);
}

```

```

while ((c = getopt_long (argc, argv, "+k:s:v", long_options, nullptr)) != -1)
{
    switch (c)
    {
        case 'k':
            kill_after = parse_duration (optarg);
            break;

        case 's':
            term_signal = operand2sig (optarg);
            if (term_signal == -1)
                usage (EXIT_CANCELED);
            break;

        case 'v':
            verbose = true;
            break;

        case FOREGROUND_OPTION:
            foreground = true;
            break;

        case PRESERVE_STATUS_OPTION:
            preserve_status = true;
            break;

        case _GETOPT_HELP_CHAR;
        case _GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

        default:
            usage (EXIT_CANCELED);
            break;
    }
}

if (argc - optind < 2)
    usage (EXIT_CANCELED);

timeout = parse_duration (argv[optind++]);

argv += optind;
command = argv[0];

/* Ensure we're in our own group so all subprocesses can be killed.
   Note we don't just put the child in a separate group as
   then we would need to worry about foreground and background groups
   and propagating signals between them. */
if (!foreground)
    setpgid (0, 0);

/* Setup handlers before fork() so that we
   handle any signals caused by child, without races. */
install_cleanup (term_signal);
signal (SIGTTIN, SIG_IGN); /* Don't stop if background child needs tty. */
signal (SIGTTOU, SIG_IGN); /* Don't stop if background child needs tty. */
install_sigchld (); /* Interrupt sigsuspend() when child exits. */

/* We configure timers so that SIGALRM is sent on expiry.
   Therefore ensure we don't inherit a mask blocking SIGALRM. */

```

```

unblock_signal (SIGALRM);

/* Block signals now, so monitored_pid is deterministic in cleanup(). */
sigset(SIG_BLOCK, &orig_set);
block_cleanup_and_chld (term_signal, &orig_set);

monitored_pid = fork ();
if (monitored_pid == -1)
{
    error (0, errno, _("fork system call failed"));
    return EXIT_CANCELED;
}
else if (monitored_pid == 0) /* child */
{
    /* Restore signal mask for child. */
    if (sigprocmask (SIG_SETMASK, &orig_set, nullptr) != 0)
    {
        error (0, errno, _("child failed to reset signal mask"));
        return EXIT_CANCELED;
    }

    /* exec doesn't reset SIG_IGN -> SIG_DFL. */
    signal (SIGTTIN, SIG_DFL);
    signal (SIGTTOU, SIG_DFL);

    execvp (argv[0], argv);

    /* exit like sh, env, nohup, ... */
    int exit_status = errno == ENOENT ? EXIT_ENOENT : EXIT_CANNOT_INVOKE;
    error (0, errno, _("failed to run command %s"), quote (command));
    return exit_status;
}
else
{
    pid_t wait_result;
    int status;

    settimeout (timeout, true);

    /* Note signals remain blocked in parent here, to ensure
       we don't cleanup() after waitpid() reaps the child,
       to avoid sending signals to a possibly different process. */

    while ((wait_result = waitpid (monitored_pid, &status, WNOHANG)) == 0)
        sigsuspend (&orig_set); /* Wait with cleanup signals unblocked. */

    if (wait_result < 0)
    {
        /* shouldn't happen. */
        error (0, errno, _("error waiting for command"));
        status = EXIT_CANCELED;
    }
    else
    {
        if (WIFEXITED (status))
            status = WEXITSTATUS (status);
        else if (WIFSIGNALED (status))
        {
            int sig = WTERMSIG (status);
            if (WCOREDUMP (status))

```

```

        error (0, 0, _("the monitored command dumped core"));
if (!timed_out && disable_core_dumps ())
{
    /* exit with the signal flag set. */
    signal (sig, SIG_DFL);
    unblock_signal (sig);
    raise (sig);
}
/* Allow users to distinguish if command was forcibly killed.
   Needed with --foreground where we don't send SIGKILL to
   the timeout process itself. */
if (timed_out && sig == SIGKILL)
    preserve_status = true;
status = sig + 128; /* what sh returns for signaled processes. */
}
else
{
    /* shouldn't happen. */
    error (0, 0, _("unknown status from command (%d)'), status);
    status = EXIT_FAILURE;
}
}

if (timed_out && !preserve_status)
    status = EXIT_TIMEDOUT;
return status;
}
"""

,
"error_category": "Conditional Compilation Issues",
"error": "Undefined Behavior or Incorrect Behavior due to Missing MVS Macro Definition",
"correct_code":
"""
/* timeout -- run a command with bounded time
Copyright (C) 2008-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */

/* timeout - Start a command, and kill it if the specified timeout expires
We try to behave like a shell starting a single (foreground) job,
and will kill the job if we receive the alarm signal we setup.
The exit status of the job is returned, or one of these errors:
EXIT_TIMEDOUT 124 job timed out
EXIT_CANCELED 125 internal error
EXIT_CANNOT_INVOKE 126 error executing job
EXIT_ENOENT 127 couldn't find job to exec

```

Caveats:

If user specifies the KILL (9) signal is to be sent on timeout,  
the monitor is killed and so exits with 128+9 rather than 124.

If you start a command in the background, which reads from the tty  
and so is immediately sent SIGTTIN to stop, then the timeout  
process will ignore this so it can timeout the command as expected.  
This can be seen with 'timeout 10 dd&' for example.

However if one brings this group to the foreground with the 'fg'  
command before the timer expires, the command will remain  
in the stop state as the shell doesn't send a SIGCONT  
because the timeout process (group leader) is already running.  
To get the command running again one can Ctrl-Z, and do fg again.  
Note one can Ctrl-C the whole job when in this state.  
I think this could be fixed but I'm not sure the extra  
complication is justified for this scenario.

Written by Pádraig Brady. \*/

```
#include <config.h>
#include <getopt.h>
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#if HAVE_PRCTL
# include <sys/prctl.h>
#endif
#include <sys/wait.h>

#include "system.h"
#include "cl-strtod.h"
#include "xstrtod.h"
#include "sig2str.h"
#include "operand2sig.h"
#include "quote.h"

#if HAVE_SETRLIMIT
/* FreeBSD 5.0 at least needs <sys/types.h> and <sys/time.h> included
before <sys/resource.h>. Currently "system.h" includes <sys/time.h>. */
# include <sys/resource.h>
#endif

/* NonStop circa 2011 lacks both SA_RESTART and siginterrupt. */
#ifndef SA_RESTART
# define SA_RESTART 0
#endif

#define PROGRAM_NAME "timeout"

#define AUTHORS proper_name_lite ("Padraig Brady", "P\303\241draig Brady")

static int timed_out;
static int term_signal = SIGTERM; /* same default as kill command. */
static pid_t monitored_pid;
static double kill_after;
static bool foreground; /* whether to use another program group. */
static bool preserve_status; /* whether to use a timeout status or not. */
static bool verbose; /* whether to diagnose timeouts or not. */
static char const *command;
```

```

/* for long options with no corresponding short option, use enum */
enum
{
    FOREGROUND_OPTION = CHAR_MAX + 1,
    PRESERVE_STATUS_OPTION
};

static struct option const long_options[] =
{
    {"kill-after", required_argument, nullptr, 'k'},
    {"signal", required_argument, nullptr, 's'},
    {"verbose", no_argument, nullptr, 'v'},
    {"foreground", no_argument, nullptr, FOREGROUND_OPTION},
    {"preserve-status", no_argument, nullptr, PRESERVE_STATUS_OPTION},
    {GETOPT_HELP_OPTION_DECL},
    {GETOPT_VERSION_OPTION_DECL},
    {nullptr, 0, nullptr, 0}
};

/* Start the timeout after which we'll receive a SIGALRM.
Round DURATION up to the next representable value.
Treat out-of-range values as if they were maximal,
as that's more useful in practice than reporting an error.
'0' means don't timeout. */
static void
settimeout (double duration, bool warn)
{
#ifndef __MVS__
#if HAVE_TIMER_SETTIME
/* timer_settime() provides potentially nanosecond resolution. */

struct timespec ts = dtotimespec (duration);
struct itimerspec its = {.it_interval = {0}, .it_value = ts};
timer_t timerid;
if (timer_create (CLOCK_REALTIME, nullptr, &timerid) == 0)
{
    if (timer_settime (timerid, 0, &its, nullptr) == 0)
        return;
    else
    {
        if (warn)
            error (0, errno, _("warning: timer_settime"));
        timer_delete (timerid);
    }
}
else if (warn && errno != ENOSYS)
    error (0, errno, _("warning: timer_create"));

#elif HAVE_SETITIMER
/* setitimer() is more portable (to Darwin for example),
but only provides microsecond resolution. */

struct timeval tv;
struct timespec ts = dtotimespec (duration);
tv.tv_sec = ts.tv_sec;
tv.tv_usec = (ts.tv_nsec + 999) / 1000;
if (tv.tv_usec == 1000 * 1000)
{
    if (tv.tv_sec != TYPE_MAXIMUM (time_t))
    {

```

```

        tv.tv_sec++;
        tv.tv_usec = 0;
    }
else
    tv.tv_usec--;
}
struct itimerval it = { .it_interval = {0}, .it_value = tv };
if (setitimer (ITIMER_REAL, &it, nullptr) == 0)
    return;
else
{
    if (warn && errno != ENOSYS)
        error (0, errno, _("warning: setitimer"));
}
#endif
#endif
/* fallback to single second resolution provided by alarm(). */

unsigned int timeint;
if (UINT_MAX <= duration)
    timeint = UINT_MAX;
else
{
    unsigned int duration_floor = duration;
    timeint = duration_floor + (duration_floor < duration);
}
alarm (timeint);
}

/* send SIG avoiding the current process. */

static int
send_sig (pid_t where, int sig)
{
/* If sending to the group, then ignore the signal,
   so we don't go into a signal loop. Note that this will ignore any of the
   signals registered in install_cleanup(), that are sent after we
   propagate the first one, which hopefully won't be an issue. Note this
   process can be implicitly multithreaded due to some timer_settime()
   implementations, therefore a signal sent to the group, can be sent
   multiple times to this process. */
if (where == 0)
    signal (sig, SIG_IGN);
return kill (where, sig);
}

/* Signal handler which is required for sigsuspend() to be interrupted
whenever SIGCHLD is received. */
static void
chld (int sig)
{
}

static void
cleanup (int sig)
{
if (sig == SIGALRM)
{
    timed_out = 1;
}

```

```

        sig = term_signal;
    }
    if (0 < monitored_pid)
    {
        if (kill_after)
        {
            int saved_errno = errno; /* settimeout may reset. */
            /* Start a new timeout after which we'll send SIGKILL. */
            term_signal = SIGKILL;
            settimeout (kill_after, false);
            kill_after = 0; /* Don't let later signals reset kill alarm. */
            errno = saved_errno;
        }

/* Send the signal directly to the monitored child,
   in case it has itself become group leader,
   or is not running in a separate group. */
if (verbose)
{
    char signature[MAX (SIG2STR_MAX, INT_BUFSIZE_BOUND (int))];
    if (sig2str (sig, signature) != 0)
        snprintf (signature, sizeof signature, "%d", sig);
    error (0, 0, _("sending signal %s to command %s"),
           signature, quote (command));
}
send_sig (monitored_pid, sig);

/* The normal case is the job has remained in our
   newly created process group, so send to all processes in that. */
if (!foreground)
{
    send_sig (0, sig);
    if (sig != SIGKILL && sig != SIGCONT)
    {
        send_sig (monitored_pid, SIGCONT);
        send_sig (0, SIGCONT);
    }
}
else if (monitored_pid == -1)
{ /* were in the parent, so let it continue to exit below. */
}
else /* monitored_pid == 0 */
{
    /* parent hasn't forked yet, or child has not exec'd yet. */
    _exit (128 + sig);
}

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    printf (_("\
Usage: %s [OPTION] DURATION COMMAND [ARG]...\n\
or: %s [OPTION]\n"), program_name, program_name);

fputs (_("\

```

```

Start COMMAND, and kill it if still running after DURATION.\n\
"), stdout);

emit_mandatory_arg_note ();

fputs (_("\
--preserve-status\n\
    exit with the same status as COMMAND, even when the\n\
    command times out\n\
--foreground\n\
    when not running timeout directly from a shell prompt,\n\
    allow COMMAND to read from the TTY and get TTY signals;\n\
    in this mode, children of COMMAND will not be timed out\n\
-k, --kill-after=DURATION\n\
    also send a KILL signal if COMMAND is still running\n\
    this long after the initial signal was sent\n\
-s, --signal=SIGNAL\n\
    specify the signal to be sent on timeout;\n\
    SIGNAL may be a name like 'HUP' or a number;\n\
    see 'kill -l' for a list of signals\n"), stdout);

fputs (_("\
-v, --verbose  diagnose to stderr any signal sent upon timeout\n"), stdout);

fputs (HELP_OPTION_DESCRIPTION, stdout);
fputs (VERSION_OPTION_DESCRIPTION, stdout);

fputs (_("\n\
DURATION is a floating point number with an optional suffix:\n\
's' for seconds (the default), 'm' for minutes, 'h' for hours or \
'd' for days.\nA duration of 0 disables the associated timeout.\n"), stdout);

fputs (_("\n\
Upon timeout, send the TERM signal to COMMAND, if no other SIGNAL specified.\n\
The TERM signal kills any process that does not block or catch that signal.\n\
It may be necessary to use the KILL signal, since this signal can't be caught.\n\
\n"), stdout);

fputs (_("\n\
Exit status:\n\
124 if COMMAND times out, and --preserve-status is not specified\n\
125 if the timeout command itself fails\n\
126 if COMMAND is found but cannot be invoked\n\
127 if COMMAND cannot be found\n\
137 if COMMAND (or timeout itself) is sent the KILL (9) signal (128+9)\n\
- the exit status of COMMAND otherwise\n\
"), stdout);

emit_ancillary_info (PROGRAM_NAME);
}

exit (status);
}

/* Given a floating point value *X, and a suffix character, SUFFIX_CHAR,
scale *X by the multiplier implied by SUFFIX_CHAR. SUFFIX_CHAR may
be the NUL byte or 's' to denote seconds, 'm' for minutes, 'h' for
hours, or 'd' for days. If SUFFIX_CHAR is invalid, don't modify *X
and return false. Otherwise return true. */

static bool
apply_time_suffix (double *x, char suffix_char)

```

```

{
int multiplier;

switch (suffix_char)
{
    case 0:
    case 's':
        multiplier = 1;
        break;
    case 'm':
        multiplier = 60;
        break;
    case 'h':
        multiplier = 60 * 60;
        break;
    case 'd':
        multiplier = 60 * 60 * 24;
        break;
    default:
        return false;
}

*x *= multiplier;

return true;
}

static double
parse_duration (char const *str)
{
double duration;
char const *ep;

if (! (xstrtod (str, &ep, &duration, cl strtod) || errno == ERANGE)
    /* Nonnegative interval. */
    || ! (0 <= duration)
    /* No extra chars after the number and an optional s,m,h,d char. */
    || (*ep && *(ep + 1))
    /* Check any suffix char and update timeout based on the suffix. */
    || !apply_time_suffix (&duration, *ep))
{
    error (0, 0, _("invalid time interval %s"), quote (str));
    usage (EXIT_CANCELED);
}

return duration;
}

static void
unblock_signal (int sig)
{
sigset_t unblock_set;
sigemptyset (&unblock_set);
sigaddset (&unblock_set, sig);
if (sigprocmask (SIG_UNBLOCK, &unblock_set, nullptr) != 0)
    error (0, errno, _("warning: sigprocmask"));
}

static void
install_sigchld (void)

```

```

{
struct sigaction sa;
sigemptyset (&sa.sa_mask); /* Allow concurrent calls to handler */
sa.sa_handler = chld;
sa.sa_flags = SA_RESTART; /* Restart syscalls if possible, as that's
                           more likely to work cleanly. */

sigaction (SIGCHLD, &sa, nullptr);

/* We inherit the signal mask from our parent process,
   so ensure SIGCHLD is not blocked. */
unblock_signal (SIGCHLD);
}

static void
install_cleanup (int sigterm)
{
struct sigaction sa;
sigemptyset (&sa.sa_mask); /* Allow concurrent calls to handler */
sa.sa_handler = cleanup;
sa.sa_flags = SA_RESTART; /* Restart syscalls if possible, as that's
                           more likely to work cleanly. */

sigaction (SIGALRM, &sa, nullptr); /* our timeout. */
sigaction (SIGINT, &sa, nullptr); /* Ctrl-C at terminal for example. */
sigaction (SIGQUIT, &sa, nullptr); /* Ctrl-\ at terminal for example. */
sigaction (SIGHUP, &sa, nullptr); /* terminal closed for example. */
sigaction (SIGTERM, &sa, nullptr); /* if killed, stop monitored proc. */
sigaction (sigterm, &sa, nullptr); /* user specified termination signal.
}

/* Block all signals which were registered with cleanup() as the signal
handler, so we never kill processes after waitpid() returns.
Also block SIGCHLD to ensure it doesn't fire between
waitpid() polling and sigsuspend() waiting for a signal.
Return original mask in OLD_SET. */
static void
block_cleanup_and_chld (int sigterm, sigset_t *old_set)
{
sigset_t block_set;
sigemptyset (&block_set);

sigaddset (&block_set, SIGALRM);
sigaddset (&block_set, SIGINT);
sigaddset (&block_set, SIGQUIT);
sigaddset (&block_set, SIGHUP);
sigaddset (&block_set, SIGTERM);
sigaddset (&block_set, sigterm);

sigaddset (&block_set, SIGCHLD);

if (sigprocmask (SIG_BLOCK, &block_set, old_set) != 0)
    error (0, errno, _("warning: sigprocmask"));
}

/* Try to disable core dumps for this process.
Return TRUE if successful, FALSE otherwise. */
static bool
disable_core_dumps (void)
{

```

```

#if HAVE_PRCTL && defined PR_SET_DUMPABLE
if (prctl (PR_SET_DUMPABLE, 0) == 0)
    return true;

#elif HAVE_SETRLIMIT && defined RLIMIT_CORE
/* Note this doesn't disable processing by a filter in
   /proc/sys/kernel/core_pattern on Linux. */
if (setrlimit (RLIMIT_CORE, &(struct rlimit) {0}) == 0)
    return true;

#else
return false;
#endif

error (0, errno, _("warning: disabling core dumps failed"));
return false;
}

int
main (int argc, char **argv)
{
double timeout;
int c;

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");

bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

initialize_exit_failure (EXIT_CANCELED);
atexit (close_stdout);

while ((c = getopt_long (argc, argv, "+k:s:v", long_options, nullptr)) != -1)
{
switch (c)
{
case 'k':
kill_after = parse_duration (optarg);
break;

case 's':
term_signal = operand2sig (optarg);
if (term_signal == -1)
    usage (EXIT_CANCELED);
break;

case 'v':
verbose = true;
break;

case FOREGROUND_OPTION:
foreground = true;
break;

case PRESERVE_STATUS_OPTION:
preserve_status = true;
break;
}
}

```

```

    case_GETOPT_HELP_CHAR;

    case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

    default:
        usage (EXIT_CANCELED);
        break;
    }
}

if (argc - optind < 2)
    usage (EXIT_CANCELED);

timeout = parse_duration (argv[optind++]);

argv += optind;
command = argv[0];

/* Ensure we're in our own group so all subprocesses can be killed.
   Note we don't just put the child in a separate group as
   then we would need to worry about foreground and background groups
   and propagating signals between them. */
if (!foreground)
    setpgid (0, 0);

/* Setup handlers before fork() so that we
   handle any signals caused by child, without races. */
install_cleanup (term_signal);
signal (SIGTTIN, SIG_IGN); /* Don't stop if background child needs tty. */
signal (SIGTTOU, SIG_IGN); /* Don't stop if background child needs tty. */
install_sigchld (); /* Interrupt sigsuspend() when child exits. */

/* We configure timers so that SIGALRM is sent on expiry.
   Therefore ensure we don't inherit a mask blocking SIGALRM. */
unblock_signal (SIGALRM);

/* Block signals now, so monitored_pid is deterministic in cleanup(). */
sigset(SIG_SETMASK, &orig_set);
block_cleanup_and_chld (term_signal, &orig_set);

monitored_pid = fork ();
if (monitored_pid == -1)
{
    error (0, errno, _("fork system call failed"));
    return EXIT_CANCELED;
}
else if (monitored_pid == 0) /* child */
{
    /* Restore signal mask for child. */
    if (sigprocmask (SIG_SETMASK, &orig_set, nullptr) != 0)
    {
        error (0, errno, _("child failed to reset signal mask"));
        return EXIT_CANCELED;
    }

    /* exec doesn't reset SIG_IGN -> SIG_DFL. */
    signal (SIGTTIN, SIG_DFL);
    signal (SIGTTOU, SIG_DFL);

    execvp (argv[0], argv);
}

```

```

/* exit like sh, env, nohup, ... */
int exit_status = errno == ENOENT ? EXIT_ENOENT : EXIT_CANNOT_INVOKE;
error (0, errno, _("failed to run command %s"), quote (command));
return exit_status;
}
else
{
pid_t wait_result;
int status;

settimeout (timeout, true);

/* Note signals remain blocked in parent here, to ensure
   we don't cleanup() after waitpid() reaps the child,
   to avoid sending signals to a possibly different process. */

while ((wait_result = waitpid (monitored_pid, &status, WNOHANG)) == 0)
  sigsuspend (&orig_set); /* Wait with cleanup signals unblocked. */

if (wait_result < 0)
{
  /* shouldn't happen. */
  error (0, errno, _("error waiting for command"));
  status = EXIT_CANCELED;
}
else
{
  if (WIFEXITED (status))
    status = WEXITSTATUS (status);
  else if (WIFSIGNALED (status))
  {
    int sig = WTERMSIG (status);
    if (WCOREDUMP (status))
      error (0, 0, _("the monitored command dumped core"));
    if (!timed_out && disable_core_dumps ())
    {
      /* exit with the signal flag set. */
      signal (sig, SIG_DFL);
      unblock_signal (sig);
      raise (sig);
    }
    /* Allow users to distinguish if command was forcibly killed.
       Needed with --foreground where we don't send SIGKILL to
       the timeout process itself. */
    if (timed_out && sig == SIGKILL)
      preserve_status = true;
    status = sig + 128; /* what sh returns for signaled processes. */
  }
  else
  {
    /* shouldn't happen. */
    error (0, 0, _("unknown status from command (%d"), status);
    status = EXIT_FAILURE;
  }
}

if (timed_out && !preserve_status)
  status = EXIT_TIMEDOUT;
return status;
}

```

```

        }
    },
    "patch":
    """
diff --git a/src/timeout.c b/src/timeout.c
index 11ae013..a0e92eb 100644
--- a/src/timeout.c
+++ b/src/timeout.c
@@ -113,7 +113,7 @@ static struct option const long_options[] =
static void
settimeout (double duration, bool warn)
{
-
+#ifndef __MVS__
#endif HAVE_TIMER_SETTIME
/* timer_settime() provides potentially nanosecond resolution. */

@@ -161,7 +161,7 @@ settimeout (double duration, bool warn)
    error (0, errno, _("warning: setitimer"));
}
#endif
-
#endif
/* fallback to single second resolution provided by alarm(). */

unsigned int timeint;
@@ -469,6 +469,7 @@ main (int argc, char **argv)
initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
+
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);
"""
},
{
    "wrong_code":
    """
/* A posix_memalign() function that works around platform bugs.
Copyright (C) 2020-2024 Free Software Foundation, Inc.

This file is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation; either version 2.1 of the
License, or (at your option) any later version.

This file is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */

#include <config.h>

/* Specification. */
#include <stdlib.h>
```

```

#include <errno.h>

int
posix_memalign (void **memptr, size_t alignment, size_t size)
#endif
{
/* Round up SIZE to the next multiple of ALIGNMENT, namely
   (SIZE + ALIGNMENT - 1) & ~(ALIGNMENT - 1). */
size += alignment - 1;
if (size >= alignment - 1) /* no overflow? */
    return posix_memalign (memptr, alignment, size & ~(size_t)(alignment - 1));
else
    return ENOMEM;
}
""",
"error_category": "Type Mismatch or Invalid Object Usage",
"error":
"""
error: called object type 'struct __notSupportedBeforeOS4V2R5__' is not a function or
function pointer
    return posix_memalign (memptr, alignment, size & ~(size_t)(alignment - 1));
~~~~~^
""",
"correct_code":
"""
/* A posix_memalign() function that works around platform bugs.
Copyright (C) 2020-2024 Free Software Foundation, Inc.

This file is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation; either version 2.1 of the
License, or (at your option) any later version.

This file is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */

```

```

#include <config.h>

/* Specification. */
#include <stdlib.h>

#include <errno.h>

int
posix_memalign (void **memptr, size_t alignment, size_t size)
#endif
{
/* FIXME: MVS does not have posix_memalign */
#ifndef __MVS__
    *memptr = malloc(size);
    return *memptr;
#else
/* Round up SIZE to the next multiple of ALIGNMENT, namely
   (SIZE + ALIGNMENT - 1) & ~(ALIGNMENT - 1). */

```

```

size += alignment - 1;
if (size >= alignment - 1) /* no overflow? */
    return posix_memalign (memptr, alignment, size & ~(size_t)(alignment - 1));
else
    return ENOMEM;
#endif
}
""",
"patch":
"""
diff --git a/lib/posix_memalign.c b/lib/posix_memalign.c
index b490665..6c10626 100644
--- a/lib/posix_memalign.c
+++ b/lib/posix_memalign.c
@@ -25,6 +25,11 @@ int
posix_memalign (void **memptr, size_t alignment, size_t size)
#undef posix_memalign
{
+/* FIXME: MVS does not have posix_memalign */
+#ifdef __MVS__
+  *memptr = malloc(size);
+  return *memptr;
+#else
/* Round up SIZE to the next multiple of ALIGNMENT, namely
   (SIZE + ALIGNMENT - 1) & ~(ALIGNMENT - 1). */
size += alignment - 1;
@@ -32,4 +37,5 @@ posix_memalign (void **memptr, size_t alignment, size_t size)
    return posix_memalign (memptr, alignment, size & ~(size_t)(alignment - 1));
else
    return ENOMEM;
#endif
}
""",
},
{
"wrong_code" :
"""
/* GNU's pinky.
Copyright (C) 1992-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>. */

/* Created by hacking who.c by Kaveh Ghazi ghazi@caip.rutgers.edu */

#include <config.h>
#include <ctype.h>
#include <getopt.h>
#include <pwd.h>

```

```

#include <stdio.h>

#include <sys/types.h>
#include "system.h"

#include "canon-host.h"
#include "hard-locale.h"
#include "readutmp.h"

/* The official name of this program (e.g., no 'g' prefix). */
#define PROGRAM_NAME "pinky"

#define AUTHORS \
proper_name ("Joseph Arceneaux"), \
proper_name ("David MacKenzie"), \
proper_name ("Kaveh Ghazi")

/* If true, display the hours:minutes since each user has touched
the keyboard, or blank if within the last minute, or days followed
by a 'd' if not within the last day. */
static bool include_idle = true;

/* If true, display a line at the top describing each field. */
static bool include_heading = true;

/* if true, display the user's full name from pw_gecos. */
static bool include_fullname = true;

/* if true, display the user's ~/.project file when doing long format. */
static bool include_project = true;

/* if true, display the user's ~/.plan file when doing long format. */
static bool include_plan = true;

/* if true, display the user's home directory and shell
when doing long format. */
static bool include_home_and_shell = true;

/* if true, use the "short" output format. */
static bool do_short_format = true;

/* If true, attempt to canonicalize hostnames via a DNS lookup. */
static bool do_lookup;

/* if true, display the ut_host field. */
#if HAVE_STRUCT_XTMP_UT_HOST
static bool include_where = true;
#endif

/* The strftime format to use for login times, and its expected
output width. */
static char const *time_format;
static int time_format_width;

/* for long options with no corresponding short option, use enum */
enum
{
LOOKUP_OPTION = CHAR_MAX + 1
};

```

```

static struct option const longopts[] =
{
    {"lookup", no_argument, nullptr, LOOKUP_OPTION},
    {GETOPT_HELP_OPTION_DECL},
    {GETOPT_VERSION_OPTION_DECL},
    {nullptr, 0, nullptr, 0}
};

/* Count and return the number of ampersands in STR. */

ATTRIBUTE PURE
static idx_t
count_ampersands (char const *str)
{
    idx_t count = 0;
    for (; *str; str++)
        count += *str == '&';
    return count;
}

/* Create a string (via xmalloc) which contains a full name by substituting
   for each ampersand in GECOS_NAME the USER_NAME string with its first
   character capitalized. The caller must ensure that GECOS_NAME contains
   no ','s. The caller also is responsible for free'ing the return value of
   this function. */

static char *
create_fullname (char const *gecos_name, char const *user_name)
{
    idx_t rsize = strlen (gecos_name) + 1;
    char *result;
    char *r;
    idx_t ampersands = count_ampersands (gecos_name);

    if (ampersands != 0)
    {
        idx_t ulen = strlen (user_name);
        ptrdiff_t product;
        if (ckd_mul (&product, ulen - 1, ampersands)
            || ckd_add (&rsize, rsize, product))
            xalloc_die ();
    }

    r = result = xmalloc (rsize);

    while (*gecos_name)
    {
        if (*gecos_name == '&')
        {
            char const *uname = user_name;
            if (islower (to_uchar (*uname)))
                *r++ = toupper (to_uchar (*uname++));
            while (*uname)
                *r++ = *uname++;
        }
        else
        {
            *r++ = *gecos_name;
        }
    }
}

```

```

        gecos_name++;
    }
*r = 0;

return result;
}

/* Return a string representing the time between WHEN and the time
that this function is first run. */

static char const *
idle_string (time_t when)
{
static time_t now = 0;
static char buf[INT_STRLEN_BOUND (intmax_t) + sizeof "d"];
time_t seconds_idle;

if (now == 0)
    time (&now);

seconds_idle = now - when;
if (seconds_idle < 60) /* One minute. */
    return " ";
if (seconds_idle < (24 * 60 * 60)) /* One day. */
{
    int hours = seconds_idle / (60 * 60);
    int minutes = (seconds_idle % (60 * 60)) / 60;
    sprintf (buf, "%02d:%02d", hours, minutes);
}
else
{
    intmax_t days = seconds_idle / (24 * 60 * 60);
    sprintf (buf, "%jdd", days);
}
return buf;
}

/* Return a time string. */
static char const *
time_string (struct gl_utmp const *utmp_ent)
{
static char buf[INT_STRLEN_BOUND (intmax_t) + sizeof "-%m-%d %H:%M"];
struct tm *tmp = localtime (&utmp_ent->ut_ts.tv_sec);

if (tmp)
{
    strftime (buf, sizeof buf, time_format, tmp);
    return buf;
}
else
    return timetostr (utmp_ent->ut_ts.tv_sec, buf);
}

/* Display a line of information about UTMP_ENT. */

static void
print_entry (struct gl_utmp const *utmp_ent)
{
struct stat stats;
time_t last_change;

```

```

char mesg;

/* If ut_line contains a space, the device name starts after the space. */
char *line = utmp_ent->ut_line;
char *space = strchr (line, ' ');
line = space ? space + 1 : line;

int dirfd;
if (IS_ABSOLUTE_FILE_NAME (line))
    dirfd = AT_FDCWD;
else
{
    static int dev_dirfd;
    if (!dev_dirfd)
    {
        dev_dirfd = open ("/dev", O_PATHSEARCH | O_DIRECTORY);
        if (dev_dirfd < 0)
            dev_dirfd = AT_FDCWD - 1;
    }
    dirfd = dev_dirfd;
}

if (AT_FDCWD <= dirfd && fstatat (dirfd, line, &stats, 0) == 0)
{
    mesg = (stats.st_mode & S_IWGRP) ? ' ' : '*';
    last_change = stats.st_atime;
}
else
{
    mesg = '?';
    last_change = 0;
}

char *ut_user = utmp_ent->ut_user;
if (strlen (ut_user, 8) < 8)
    printf ("%-8s", ut_user);
else
    fputs (ut_user, stdout);

if (include_fullname)
{
    struct passwd *pw = getpwnam (ut_user);
    if (pw == nullptr)
        /* TRANSLATORS: Real name is unknown; at most 19 characters. */
        printf (" %19s", _("      ???"));
    else
    {
        char *const comma = strchr (pw->pw_gecos, ',');
        char *result;

        if (comma)
            *comma = '\0';

        result = create_fullname (pw->pw_gecos, pw->pw_name);
        printf (" %-19.19s", result);
        free (result);
    }
}
fputc (' ', stdout);

```

```

fputc (mesg, stdout);
if (strnlen (utmp_ent->ut_line, 8) < 8)
    printf ("% -8s", utmp_ent->ut_line);
else
    fputs (utmp_ent->ut_line, stdout);

if (include_idle)
{
    if (last_change)
        printf (" % -6s", idle_string (last_change));
    else
        /* TRANSLATORS: Idle time is unknown; at most 5 characters. */
        printf (" % -6s", _("?????"));
}

printf (" %s", time_string (utmp_ent));

#endif HAVE_STRUCT_XTMP_UT_HOST
if (include_where && utmp_ent->ut_host[0])
{
    char *host = nullptr;
    char *display = nullptr;
    char *ut_host = utmp_ent->ut_host;

    /* Look for an X display. */
    display = strchr (ut_host, ':');
    if (display)
        *display++ = '\0';

    if (*ut_host && do_lookup)
        /* See if we can canonicalize it. */
        host = canon_host (ut_host);
    if (!host)
        host = ut_host;

    fputc (' ', stdout);
    fputs (host, stdout);
    if (display)
    {
        fputc (':', stdout);
        fputs (display, stdout);
    }

    if (host != ut_host)
        free (host);
}
#endif
putchar ('\n');

/* Display a verbose line of information about UTMP_ENT. */

static void
print_long_entry (const char name[])
{
    struct passwd *pw;

    pw = getpwnam (name);
}

```

```

printf (_("Login name: "));
printf ("%-28s", name);

printf (_("In real life: "));
if (pw == nullptr)
{
    /* TRANSLATORS: Real name is unknown; no hard limit. */
    printf (" %s", _("???\\n"));
    return;
}
else
{
    char *const comma = strchr (pw->pw_gecos, ',');
    char *result;

    if (comma)
        *comma = '\\0';

    result = create_fullname (pw->pw_gecos, pw->pw_name);
    printf (" %s", result);
    free (result);
}

putchar ('\\n');

if (include_home_and_shell)
{
    printf (_("Directory: "));
    printf ("%-29s", pw->pw_dir);
    printf (_("Shell: "));
    printf (" %s", pw->pw_shell);
    putchar ('\\n');
}

if (include_project)
{
    FILE *stream;
    char buf[1024];
    char const *const baseproject = "./.project";
    char *const project =
        xmalloc (strlen (pw->pw_dir) + strlen (baseproject) + 1);
    stpcpy (stpcpy (project, pw->pw_dir), baseproject);

    stream = fopen (project, "r");
    if (stream)
    {
        size_t bytes;

        printf (_("Project: "));

        while ((bytes = fread (buf, 1, sizeof (buf), stream)) > 0)
            fwrite (buf, 1, bytes, stdout);
        fclose (stream);
    }

    free (project);
}

if (include_plan)
{

```

```

FILE *stream;
char buf[1024];
char const *const baseplan = "./.plan";
char *const plan =
    xmalloc (strlen (pw->pw_dir) + strlen (baseplan) + 1);
stpcpy (stpcpy (plan, pw->pw_dir), baseplan);

stream = fopen (plan, "r");
if (stream)
{
    size_t bytes;

    printf (_("Plan:\n"));

    while ((bytes = fread (buf, 1, sizeof (buf), stream)) > 0)
        fwrite (buf, 1, bytes, stdout);
    fclose (stream);
}

free (plan);
}

putchar ('\n');
}

/* Print the username of each valid entry and the number of valid entries
in UTMP_BUF, which should have N elements. */

static void
print_heading (void)
{
printf ("%-8s", _("Login"));
if (include_fullname)
    printf (" %-19s", _("Name"));
printf (" %-9s", _("TTY"));
if (include_idle)
    printf (" %-6s", _("Idle"));
printf (" %-*s", time_format_width, _("When"));
#ifndef HAVE_STRUCT_XTMP_UT_HOST
if (include_where)
    printf (" %s", _("Where"));
#endif
putchar ('\n');
}

/* Display UTMP_BUF, which should have N entries. */

static void
scan_entries (idx_t n, struct gl_utmp const *utmp_buf,
              const int argc_names, char *const argv_names[])
{
if (hard_locale (LC_TIME))
{
    time_format = "%Y-%m-%d %H:%M";
    time_format_width = 4 + 1 + 2 + 1 + 2 + 1 + 2 + 1 + 2;
}
else
{
    time_format = "%b %e %H:%M";
    time_format_width = 3 + 1 + 2 + 1 + 2 + 1 + 2;
}

```

```

    }

if (include_heading)
    print_heading ();

while (n--)
{
    if (IS_USER_PROCESS (utmp_buf))
    {
        if (argc_names)
        {
            for (int i = 0; i < argc_names; i++)
                if (STREQ (utmp_buf->ut_user, argv_names[i]))
                {
                    print_entry (utmp_buf);
                    break;
                }
        }
        else
            print_entry (utmp_buf);
    }
    utmp_buf++;
}
}

/* Display a list of who is on the system, according to utmp file FILENAME. */

static void
short_pinky (char const *filename,
             const int argc_names, char *const argv_names[])
{
idx_t n_users;
struct gl_utmp *utmp_buf;
if (read_utmp (filename, &n_users, &utmp_buf, READ_UTMP_USER_PROCESS) != 0)
    error (EXIT_FAILURE, errno, "%s", quotef (filename));

scan_entries (n_users, utmp_buf, argc_names, argv_names);
exit (EXIT_SUCCESS);
}

static void
long_pinky (const int argc_names, char *const argv_names[])
{
for (int i = 0; i < argc_names; i++)
    print_long_entry (argv_names[i]);
}

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    {
        printf (_("Usage: %s [OPTION]... [USER]...\n"), program_name);
        fputs (_("\
\n\
-l      produce long format output for the specified USERs\n\
-b      omit the user's home directory and shell in long format\n\
-h      omit the user's project file in long format\n\
"),

```

```

-p      omit the user's plan file in long format\n\
-s      do short format output, this is the default\n\
"), stdout);
    fputs (_("\
-f      omit the line of column headings in short format\n\
-w      omit the user's full name in short format\n\
-i      omit the user's full name and remote host in short format\n\
-q      omit the user's full name, remote host and idle time\n\
      in short format\n\
"), stdout);
    fputs (_("\
--lookup  attempt to canonicalize hostnames via DNS\n\
"), stdout);
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
    printf (_("\
\n\
A lightweight 'finger' program; print user information.\n\
The utmp file will be %s.\n\
"), UTMP_FILE);
    emit_ancillary_info (PROGRAM_NAME);
}
exit (status);
}

int
main (int argc, char **argv)
{
int optc;
int n_users;

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

atexit (close_stdout);

while ((optc = getopt_long (argc, argv, "sfwiqbhlp", longopts, nullptr))
!= -1)
{
switch (optc)
{
case 's':
do_short_format = true;
break;

case 'I':
do_short_format = false;
break;

case 'f':
include_heading = false;
break;

case 'w':
include_fullname = false;
break;
}
}

```

```

        case 'i':
            include_fullname = false;
#ifndef HAVE_STRUCT_UTMP_UT_HOST
            include_where = false;
#endif
            break;

        case 'q':
            include_fullname = false;
#ifndef HAVE_STRUCT_UTMP_UT_HOST
            include_where = false;
#endif
            break;
            include_idle = false;
            break;

        case 'h':
            include_project = false;
            break;

        case 'p':
            include_plan = false;
            break;

        case 'b':
            include_home_and_shell = false;
            break;

        case LOOKUP_OPTION:
            do_lookup = true;
            break;

        case getopt_HELP_CHAR;

        case getopt_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

        default:
            usage (EXIT_FAILURE);
        }
    }

n_users = argc - optind;

if (!do_short_format && n_users == 0)
{
    error (0, 0, _("no username specified; at least one must be\
specified when using -l"));
    usage (EXIT_FAILURE);
}

if (do_short_format)
    short_pinky (UTMP_FILE, n_users, argv + optind);
else
    long_pinky (n_users, argv + optind);

return EXIT_SUCCESS;
}
"""

,
"error_category": "Type Mismatch and Missing or Incompatible Struct Member",
"error": """

```

```
warning: incompatible pointer types passing 'struct stat *' to parameter of type 'struct stat
*' [-Wincompatible-pointer-types]
    if (AT_FDCWD <= dirfd && fstatat (dirfd, line, &stats, 0) == 0)

note: passing argument to parameter 'st' here
      (int fd, char const *restrict name, struct stat *restrict st,
error: no member named 'pw_gecos' in 'struct passwd'
      char *const comma = strchr (pw->pw_gecos, ',');
      ~~~ ^
error: no member named 'pw_gecos' in 'struct passwd'
      result = create_fullname (pw->pw_gecos, pw->pw_name);
      ~~~ ^
error: no member named 'pw_gecos' in 'struct passwd'
      char *const comma = strchr (pw->pw_gecos, ',');
      ~~~ ^
error: no member named 'pw_gecos' in 'struct passwd'
      result = create_fullname (pw->pw_gecos, pw->pw_name);
      ~~~ ^
"""
"correct_code": """
/* GNU's pinky.
Copyright (C) 1992-2024 Free Software Foundation, Inc.
```

This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License  
along with this program. If not, see <<https://www.gnu.org/licenses/>>. \*/

```
/* Created by hacking who.c by Kaveh Ghazi ghazi@caip.rutgers.edu */
```

```
#include <config.h>
#include <ctype.h>
#include <getopt.h>
#include <pwd.h>
#include <stdio.h>

#include <sys/types.h>
#include "system.h"

#include "canon-host.h"
#include "hard-locale.h"
#include "readutmp.h"

/* The official name of this program (e.g., no 'g' prefix). */
#define PROGRAM_NAME "pinky"

#define AUTHORS \
proper_name ("Joseph Arceneaux"), \
proper_name ("David MacKenzie"), \
proper_name ("Kaveh Ghazi")
```

```

/* If true, display the hours:minutes since each user has touched
the keyboard, or blank if within the last minute, or days followed
by a 'd' if not within the last day. */
static bool include_idle = true;

/* If true, display a line at the top describing each field. */
static bool include_heading = true;

/* if true, display the user's full name from pw_gecos. */
static bool include_fullname = true;

/* if true, display the user's ~/.project file when doing long format. */
static bool include_project = true;

/* if true, display the user's ~/.plan file when doing long format. */
static bool include_plan = true;

/* if true, display the user's home directory and shell
when doing long format. */
static bool include_home_and_shell = true;

/* if true, use the "short" output format. */
static bool do_short_format = true;

/* If true, attempt to canonicalize hostnames via a DNS lookup. */
static bool do_lookup;

/* if true, display the ut_host field. */
#ifndef HAVE_STRUCT_XTMP_UT_HOST
static bool include_where = true;
#endif

/* The strftime format to use for login times, and its expected
output width. */
static char const *time_format;
static int time_format_width;

/* for long options with no corresponding short option, use enum */
enum
{
    LOOKUP_OPTION = CHAR_MAX + 1
};

static struct option const longopts[] =
{
    {"lookup", no_argument, nullptr, LOOKUP_OPTION},
    {GETOPT_HELP_OPTION_DECL},
    {GETOPT_VERSION_OPTION_DECL},
    {nullptr, 0, nullptr, 0}
};

/* Count and return the number of ampersands in STR. */

ATTRIBUTE_PURE
static idx_t
count_ampersands (char const *str)
{
    idx_t count = 0;
    for (; *str; str++)

```

```

        count += *str == '&';
    return count;
}

/* Create a string (via xmalloc) which contains a full name by substituting
for each ampersand in GECOS_NAME the USER_NAME string with its first
character capitalized. The caller must ensure that GECOS_NAME contains
no ','s. The caller also is responsible for free'ing the return value of
this function. */

static char *
create_fullname (char const *gecos_name, char const *user_name)
{
    idx_t rsize = strlen (gecos_name) + 1;
    char *result;
    char *r;
    idx_t ampersands = count_ampersands (gecos_name);

    if (ampersands != 0)
    {
        idx_t ulen = strlen (user_name);
        ptrdiff_t product;
        if (ckd_mul (&product, ulen - 1, ampersands)
            || ckd_add (&rsize, rsize, product))
            xalloc_die ();
    }

    r = result = xmalloc (rsize);

    while (*gecos_name)
    {
        if (*gecos_name == '&')
        {
            char const *uname = user_name;
            if (islower (to_uchar (*uname)))
                *r++ = toupper (to_uchar (*uname++));
            while (*uname)
                *r++ = *uname++;
        }
        else
        {
            *r++ = *gecos_name;
        }

        gecos_name++;
    }
    *r = 0;

    return result;
}

/* Return a string representing the time between WHEN and the time
that this function is first run. */

static char const *
idle_string (time_t when)
{
    static time_t now = 0;
    static char buf[INT_STRLEN_BOUND (intmax_t) + sizeof "d"];
    time_t seconds_idle;

```

```

if (now == 0)
    time (&now);

seconds_idle = now - when;
if (seconds_idle < 60) /* One minute. */
    return "";
if (seconds_idle < (24 * 60 * 60))      /* One day. */
{
    int hours = seconds_idle / (60 * 60);
    int minutes = (seconds_idle % (60 * 60)) / 60;
    sprintf (buf, "%02d:%02d", hours, minutes);
}
else
{
    intmax_t days = seconds_idle / (24 * 60 * 60);
    sprintf (buf, "%jdd", days);
}
return buf;
}

/* Return a time string. */
static char const *
time_string (struct gl_utmp const *utmp_ent)
{
static char buf[INT_STRLEN_BOUND (intmax_t) + sizeof "-%m-%d %H:%M"];
struct tm *tmp = localtime (&utmp_ent->ut_ts.tv_sec);

if (tmp)
{
    strftime (buf, sizeof buf, time_format, tmp);
    return buf;
}
else
    return timetostr (utmp_ent->ut_ts.tv_sec, buf);
}

/* Display a line of information about UTMP_ENT. */

static void
print_entry (struct gl_utmp const *utmp_ent)
{
struct stat stats;
time_t last_change;
char mesg;

/* If ut_line contains a space, the device name starts after the space. */
char *line = utmp_ent->ut_line;
char *space = strchr (line, ' ');
line = space ? space + 1 : line;

int dirfd;
if (IS_ABSOLUTE_FILE_NAME (line))
    dirfd = AT_FDCWD;
else
{
    static int dev_dirfd;
    if (!dev_dirfd)
    {
        dev_dirfd = open ("/dev", O_PATHSEARCH | O_DIRECTORY);
}
}
}

```

```

        if (dev_dirfd < 0)
            dev_dirfd = AT_FDCWD - 1;
    }
    dirfd = dev_dirfd;
}

if (AT_FDCWD <= dirfd && fstatat (dirfd, line, &stats, 0) == 0)
{
    mesg = (stats.st_mode & S_IWGRP) ? ' ' : '*';
    last_change = stats.st_atime;
}
else
{
    mesg = '?';
    last_change = 0;
}

char *ut_user = utmp_ent->ut_user;
if (strnlen (ut_user, 8) < 8)
    printf ("%-8s", ut_user);
else
    fputs (ut_user, stdout);

if (include_fullname)
{
    struct passwd *pw = getpwnam (ut_user);
    if (pw == nullptr)
        /* TRANSLATORS: Real name is unknown; at most 19 characters. */
        printf (" %19s", _("      ???"));
    else
    {
        char *result;
#ifndef __MVS__
        char *const comma = strchr (pw->pw_gecos, ',');
        if (comma)
            *comma = '\0';
#endif
# ifdef __MVS__
        result = create_fullname ("", pw->pw_name);
# else
        result = create_fullname (pw->pw_gecos, pw->pw_name);
# endif
        printf (" %-19.19s", result);
        free (result);
    }
}

fputc (' ', stdout);
fputc (mesg, stdout);
if (strnlen (utmp_ent->ut_line, 8) < 8)
    printf ("%-8s", utmp_ent->ut_line);
else
    fputs (utmp_ent->ut_line, stdout);

if (include_idle)
{
    if (last_change)
        printf (" %-6s", idle_string (last_change));
}

```

```

else
    /* TRANSLATORS: Idle time is unknown; at most 5 characters. */
    printf (" % -6s", _("?????"));
}

printf (" %s", time_string (utmp_ent));

#endif HAVE_STRUCT_UTMP_UT_HOST
if (include_where && utmp_ent->ut_host[0])
{
    char *host = nullptr;
    char *display = nullptr;
    char *ut_host = utmp_ent->ut_host;

    /* Look for an X display. */
    display = strchr (ut_host, ':');
    if (display)
        *display++ = '\0';

    if (*ut_host && do_lookup)
        /* See if we can canonicalize it. */
        host = canon_host (ut_host);
    if (! host)
        host = ut_host;

    fputc (' ', stdout);
    fputs (host, stdout);
    if (display)
    {
        fputc (':', stdout);
        fputs (display, stdout);
    }

    if (host != ut_host)
        free (host);
}
#endif

putchar ('\n');

/* Display a verbose line of information about UTMP_ENT. */

static void
print_long_entry (const char name[])
{
    struct passwd *pw;

    pw = getpwnam (name);

    printf (_("Login name: "));
    printf ("% -28s", name);

    printf (_("In real life: "));
    if (pw == nullptr)
    {
        /* TRANSLATORS: Real name is unknown; no hard limit. */
        printf (" %s", _("???\\n"));
        return;
    }
}

```

```

else
{
    char *result;
#ifndef __MVS__
    char *const comma = strchr (pw->pw_gecos, ',');
    if (comma)
        *comma = '\0';
#endif

#ifndef __MVS__
    result = create_fullname ("", pw->pw_name);
#else
    result = create_fullname (pw->pw_gecos, pw->pw_name);
#endif
    printf (" %s", result);
    free (result);
}

putchar ('\n');

if (include_home_and_shell)
{
    printf (_("Directory: "));
    printf ("%-29s", pw->pw_dir);
    printf (_("Shell: "));
    printf ("%s", pw->pw_shell);
    putchar ('\n');
}

if (include_project)
{
    FILE *stream;
    char buf[1024];
    char const *const baseproject = "./.project";
    char *const project =
        xmalloc (strlen (pw->pw_dir) + strlen (baseproject) + 1);
    stpcpy (stpcpy (project, pw->pw_dir), baseproject);

    stream = fopen (project, "r");
    if (stream)
    {
        size_t bytes;

        printf (_("Project: "));

        while ((bytes = fread (buf, 1, sizeof (buf), stream)) > 0)
            fwrite (buf, 1, bytes, stdout);
        fclose (stream);
    }

    free (project);
}

if (include_plan)
{
    FILE *stream;
    char buf[1024];
    char const *const baseplan = "./.plan";
    char *const plan =

```

```

xmalloc (strlen (pw->pw_dir) + strlen (baseplan) + 1);
stpcpy (stpcpy (plan, pw->pw_dir), baseplan);

stream = fopen (plan, "r");
if (stream)
{
    size_t bytes;

    printf (_("Plan:\n"));

    while ((bytes = fread (buf, 1, sizeof (buf), stream)) > 0)
        fwrite (buf, 1, bytes, stdout);
    fclose (stream);
}

free (plan);
}

putchar ('\n');

/* Print the username of each valid entry and the number of valid entries
in UTMP_BUF, which should have N elements. */

static void
print_heading (void)
{
printf ("%-8s", _("Login"));
if (include_fullname)
    printf ("%-19s", _("Name"));
printf ("%-9s", _("TTY"));
if (include_idle)
    printf ("%-6s", _("Idle"));
printf ("%-*s", time_format_width, _("When"));
#ifndef HAVE_STRUCT_XTMP_UT_HOST
if (include_where)
    printf ("%s", _("Where"));
#endif
putchar ('\n');
}

/* Display UTMP_BUF, which should have N entries. */

static void
scan_entries (idx_t n, struct gl_utmp const *utmp_buf,
              const int argc_names, char *const argv_names[])
{
if (hard_locale (LC_TIME))
{
    time_format = "%Y-%m-%d %H:%M";
    time_format_width = 4 + 1 + 2 + 1 + 2 + 1 + 2 + 1 + 2;
}
else
{
    time_format = "%b %e %H:%M";
    time_format_width = 3 + 1 + 2 + 1 + 2 + 1 + 2;
}

if (include_heading)
    print_heading ();

```

```

while (n--)
{
    if (IS_USER_PROCESS (utmp_buf))
    {
        if (argc_names)
        {
            for (int i = 0; i < argc_names; i++)
                if (STREQ (utmp_buf->ut_user, argv_names[i]))
                {
                    print_entry (utmp_buf);
                    break;
                }
        }
        else
            print_entry (utmp_buf);
    }
    utmp_buf++;
}
}

/* Display a list of who is on the system, according to utmp file FILENAME. */

static void
short_pinky (char const *filename,
             const int argc_names, char *const argv_names[])
{
idx_t n_users;
struct gl_utmp *utmp_buf;
if (read_utmp (filename, &n_users, &utmp_buf, READ_UTMP_USER_PROCESS) != 0)
    error (EXIT_FAILURE, errno, "%s", quotef (filename));

scan_entries (n_users, utmp_buf, argc_names, argv_names);
exit (EXIT_SUCCESS);
}

static void
long_pinky (const int argc_names, char *const argv_names[])
{
for (int i = 0; i < argc_names; i++)
    print_long_entry (argv_names[i]);
}

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    printf (_("Usage: %s [OPTION]... [USER]...\n"), program_name);
    fputs (_("\
\n\
-l      produce long format output for the specified USERs\n\
-b      omit the user's home directory and shell in long format\n\
-h      omit the user's project file in long format\n\
-p      omit the user's plan file in long format\n\
-s      do short format output, this is the default\n\
"), stdout);
    fputs (_("\

```

```

-f      omit the line of column headings in short format\n\
-w      omit the user's full name in short format\n\
-i      omit the user's full name and remote host in short format\n\
-q      omit the user's full name, remote host and idle time\n\
       in short format\n\
"), stdout);
    fputs (_("\
--lookup  attempt to canonicalize hostnames via DNS\n\
"), stdout);
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);
    printf (_("\
\n\
A lightweight 'finger' program; print user information.\n\
The utmp file will be %s.\n\
"), UTMP_FILE);
    emit_ancillary_info (PROGRAM_NAME);
}
exit (status);
}

int
main (int argc, char **argv)
{
int optc;
int n_users;

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

atexit (close_stdout);

while ((optc = getopt_long (argc, argv, "sfwiqbhlp", longopts, nullptr))
!= -1)
{
switch (optc)
{
case 's':
do_short_format = true;
break;

case 'I':
do_short_format = false;
break;

case 'f':
include_heading = false;
break;

case 'w':
include_fullname = false;
break;

case 'i':
include_fullname = false;
#endif HAVE_STRUCT_UTMP_UT_HOST
include_where = false;
}

```

```

#endif
    break;

    case 'q':
        include_fullname = false;
#ifndef HAVE_STRUCT_XTMP_UT_HOST
        include_where = false;
#endif
        include_idle = false;
        break;

    case 'h':
        include_project = false;
        break;

    case 'p':
        include_plan = false;
        break;

    case 'b':
        include_home_and_shell = false;
        break;

    case LOOKUP_OPTION:
        do_lookup = true;
        break;

    case getopt_HELP_CHAR;

    case getopt_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

    default:
        usage (EXIT_FAILURE);
    }
}

n_users = argc - optind;

if (!do_short_format && n_users == 0)
{
    error (0, 0, _("no username specified; at least one must be\
specified when using -l"));
    usage (EXIT_FAILURE);
}

if (do_short_format)
    short_pinky (UTMP_FILE, n_users, argv + optind);
else
    long_pinky (n_users, argv + optind);

return EXIT_SUCCESS;
}
"""

,
"patch": """

diff --git a/src/pinky.c b/src/pinky.c
index 88862c2..39842df 100644
--- a/src/pinky.c
+++ b/src/pinky.c
@@ -242,13 +242,19 @@ print_entry (const STRUCT_UTMP *utmp_ent)

```

```

        printf ("%19s", _("      ???"));
    else
    {
-     char *const comma = strchr (pw->pw_gecos, ',');
     char *result;
+ #ifndef __MVS__
+     char *const comma = strchr (pw->pw_gecos, ',');

     if (comma)
         *comma = '\0';
+ #endif

+ #ifdef __MVS__
+     result = create_fullname ("", pw->pw_name);
+ #else
     result = create_fullname (pw->pw_gecos, pw->pw_name);
+ #endif
     printf ("%19s", result);
     free (result);
    }
@@ -323,13 +329,19 @@ print_long_entry (const char name[])
}
else
{
-     char *const comma = strchr (pw->pw_gecos, ',');
-     char *result;
+     char *result;
+ #ifndef __MVS__
+     char *const comma = strchr (pw->pw_gecos, ',');

-     if (comma)
-         *comma = '\0';
+     if (comma)
+         *comma = '\0';
+ #endif

-     result = create_fullname (pw->pw_gecos, pw->pw_name);
+ #ifdef __MVS__
+     result = create_fullname ("", pw->pw_name);
+ #else
+     result = create_fullname (pw->pw_gecos, pw->pw_name);
+ #endif
     printf ("%s", result);
     free (result);
    }
    """
},
{
  "wrong_code": """
/* stat.c -- display file or file system status
Copyright (C) 2001-2024 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,

```

This program is free software: you can redistribute it and/or modify  
 it under the terms of the GNU General Public License as published by  
 the Free Software Foundation, either version 3 of the License, or  
 (at your option) any later version.

This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Written by Michael Meskes. \*/

```
#include <config.h>

/* Keep this conditional in sync with the similar conditional in
..../m4/stat-prog.m4. */
#ifndef __USE_MINGW32_CRT
# if ((STAT_STATVFS || STAT_STATVFS64) \
     && (HAVE_STRUCT_STATVFS_F_BASETYPE || \
HAVE_STRUCT_STATVFS_F_FSTYPENAME \
     || (! HAVE_STRUCT_STATFS_F_FSTYPENAME && \
HAVE_STRUCT_STATVFS_F_TYPE)))
# define USE_STATVFS 1
#else
# define USE_STATVFS 0
#endif

#include <stdio.h>
#include <sys/types.h>
#include <pwd.h>
#include <grp.h>
#if USE_STATVFS
# include <sys/statvfs.h>
#elif HAVE_SYS_VFS_H
# include <sys/vfs.h>
#elif HAVE_SYS_MOUNT_H && HAVE_SYS_PARAM_H
/* NOTE: freebsd5.0 needs sys/param.h and sys/mount.h for statfs.
It does have statvfs.h, but shouldn't use it, since it doesn't
HAVE_STRUCT_STATVFS_F_BASETYPE. So find a clean way to fix it. */
/* NetBSD 1.5.2 needs these, for the declaration of struct statfs. */
# include <sys/param.h>
# include <sys/mount.h>
# if HAVE_NFS_NFS_CLNT_H && HAVE_NFS_VFS_H
/* Ultrix 4.4 needs these for the declaration of struct statfs. */
# include <netinet/in.h>
# include <nfs/nfs_clnt.h>
# include <nfs/vfs.h>
#endif
#elif HAVE_OS_H /* BeOS */
# include <fs_info.h>
#endif
#include <selinux/selinux.h>
#include <getopt.h>

#include "system.h"

#include "areadlink.h"
#include "argmatch.h"
#include "c-ctype.h"
#include "file-type.h"
#include "filemode.h"
#include "fs.h"
#include "mountlist.h"
#include "quote.h"
```

```

#include "stat-size.h"
#include "stat-time.h"
#include "strftime.h"
#include "find-mount-point.h"
#include "xvasprintf.h"
#include "statx.h"

#if HAVE_STATX && defined STATX_INO
# define USE_STATX 1
#else
# define USE_STATX 0
#endif

#if USE_STATVFS
# define STRUCT_STATXFS_F_FSID_IS_INTEGER
STRUCT_STATVFS_F_FSID_IS_INTEGER
# define HAVE_STRUCT_STATXFS_F_TYPE HAVE_STRUCT_STATVFS_F_TYPE
# if HAVE_STRUCT_STATVFS_F_NAMEMAX
# define SB_F_NAMEMAX(S) ((S)->f_namemax)
# endif
# if !STAT_STATVFS && STAT_STATVFS64
# define STRUCT_STATVFS struct statvfs64
# define STATFS statvfs64
# else
# define STRUCT_STATVFS struct statvfs
# define STATFS statvfs
# endif
# define STATFS_FRSIZE(S) ((S)->f_frsize)
#else
# define HAVE_STRUCT_STATXFS_F_TYPE HAVE_STRUCT_STATFS_F_TYPE
# if HAVE_STRUCT_STATFS_F_NAMELEN
# define SB_F_NAMEMAX(S) ((S)->f_namelen)
# elif HAVE_STRUCT_STATFS_F_NAMEMAX
# define SB_F_NAMEMAX(S) ((S)->f_namemax)
# endif
# define STATFS statfs
# if HAVE_OS_H /* BeOS */
/* BeOS has a statvfs function, but it does not return sensible values
for f_files, f_ffree and f_favail, and lacks f_type, f_basetype and
f_fstypename. Use 'struct fs_info' instead. */
NODISCARD
static int
statfs (char const *filename, struct fs_info *buf)
{
dev_t device = dev_for_path (filename);
if (device < 0)
{
errno = (device == B_ENTRY_NOT_FOUND ? ENOENT
: device == B_BAD_VALUE ? EINVAL
: device == B_NAME_TOO_LONG ? ENAMETOOLONG
: device == B_NO_MEMORY ? ENOMEM
: device == B_FILE_ERROR ? EIO
: 0);
return -1;
}
/* If successful, buf->dev will be == device. */
return fs_stat_dev (device, buf);
}
# define f_fsid dev
# define f_blocks total_blocks

```

```

# define f_bfree free_blocks
# define f_bavail free_blocks
# define f_bsize io_size
# define f_files total_nodes
# define f_ffree free_nodes
# define STRUCT_STATVFS struct fs_info
# define STRUCT_STATXFS_F_FSID_IS_INTEGER true
# define STATFS_FRSIZE(S) ((S)->block_size)
# else
# define STRUCT_STATVFS struct statfs
# define STRUCT_STATXFS_F_FSID_IS_INTEGER
STRUCT_STATFS_F_FSID_IS_INTEGER
# if HAVE_STRUCT_STATFS_F_FRSIZE
# define STATFS_FRSIZE(S) ((S)->f_frsize)
# else
# define STATFS_FRSIZE(S) 0
# endif
# endif
#endif

#ifndef SB_F_NAMEMAX
#define OUT_NAMEMAX out_uint
#else
/* Depending on whether statvfs or statfs is used,
neither f_namemax or f_namelen may be available. */
#define SB_F_NAMEMAX(S) "?"
#define OUT_NAMEMAX out_string
#endif

#if HAVE_STRUCT_STATVFS_F_BASETYPE
#define STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME f_basetype
#else
#define HAVE_STRUCT_STATVFS_F_FSTYPENAME ||
HAVE_STRUCT_STATFS_F_FSTYPENAME
#define STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME f_fstypename
#define HAVE_OS_H /* BeOS */
#define STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME fsh_name
#endif
#endif

#if HAVE_GETATTRAT
#include <attr.h>
#include <sys/nvpair.h>
#endif

/* FIXME: these are used by printf.c, too */
#define isodigit(c) ('0' <= (c) && (c) <= '7')
#define octtobin(c) ((c) - '0')
#define hextobin(c) ((c) >= 'a' && (c) <= 'f' ? (c) - 'a' + 10 : \
(c) >= 'A' && (c) <= 'F' ? (c) - 'A' + 10 : (c) - '0')

static char const digits[] = "0123456789";

/* Flags that are portable for use in printf, for at least one
conversion specifier; make_format removes non-portable flags as
needed for particular specifiers. The glibc 2.2 extension "l" is
listed here; it is removed by make_format because it has undefined
behavior elsewhere and because it is incompatible with
out_epoch_sec. */
static char const printf_flags[] = "'-+ #0l";

```

```

/* Formats for the --terse option. */
static char const fmt_terse_fs[] = "%n %i %l %t %s %S %b %f %a %c %d\n";
static char const fmt_terse_regular[] = "%n %s %b %f %u %g %D %i %h %t %T"
    " %X %Y %Z %W %o\n";
static char const fmt_terse_selinux[] = "%n %s %b %f %u %g %D %i %h %t %T"
    " %X %Y %Z %W %o %C\n";

#define PROGRAM_NAME "stat"

#define AUTHORS proper_name ("Michael Meskes")

enum
{
PRINTF_OPTION = CHAR_MAX + 1
};

enum cached_mode
{
cached_default,
cached_never,
cached_always
};

static char const *const cached_args[] =
{
"default", "never", "always", nullptr
};

static enum cached_mode const cached_modes[] =
{
cached_default, cached_never, cached_always
};

static struct option const long_options[] =
{
{"dereference", no_argument, nullptr, 'L'},
 {"file-system", no_argument, nullptr, 'f'},
 {"format", required_argument, nullptr, 'c'},
 {"printf", required_argument, nullptr, PRINTF_OPTION},
 {"terse", no_argument, nullptr, 't'},
 {"cached", required_argument, nullptr, 0},
 {GETOPT_HELP_OPTION_DECL},
 {GETOPT_VERSION_OPTION_DECL},
 {nullptr, 0, nullptr, 0}
};

/* Whether to follow symbolic links; True for --dereference (-L). */
static bool follow_links;

/* Whether to interpret backslash-escape sequences.
True for --printf=FMT, not for --format=FMT (-c). */
static bool interpret_backslash_escapes;

/* The trailing delimiter string:
"" for --printf=FMT, "\n" for --format=FMT (-c). */
static char const *trailing_delim = "";

/* The representation of the decimal point in the current locale. */
static char const *decimal_point;

```

```

static size_t decimal_point_len;

static bool
print_stat (char *pformat, size_t prefix_len, char mod, char m,
            int fd, char const *filename, void const *data);

/* Return the type of the specified file system.
Some systems have statvfs.f_basetype[FSTYPSZ] (AIX, HP-UX, and Solaris).
Others have statvfs.f_fstypename[_VFS_NAMELEN] (NetBSD 3.0).
Others have statfs.f_fstypename[MFSNAMELEN] (NetBSD 1.5.2).
Still others have neither and have to get by with f_type (GNU/Linux).
But f_type may only exist in statfs (Cygwin). */
NODISCARD
static char const *
human_fstype (STRUCT_STATVFS const *statfsbuf)
{
#ifndef STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME
return statfsbuf->STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME;
#else
switch (statfsbuf->f_type)
{
# if defined __linux__ || defined __ANDROID__

/* Compare with what's in libc:
f=/a/libc/sysdeps/unix/sysv/linux/linux_fsinfo.h
sed -n '/ADFS_SUPER_MAGIC/,/SYSFS_MAGIC/p' $f \
| perl -n -e '/#define (.*)_(?:SUPER_)MAGIC\s+0x(\S+)/' \
-e 'and print "case S_MAGIC_$1: \\\n\t0x". uc($2) . "\\\n"' \
| sort > sym_libc
perl -ne '/^\s+(case S_MAGIC_.*?): \\\n\t0x(\S+)\\\\V/' \
-e 'and do { $v=uc$2; print "$1: \\\n\t0x$v \\\n"}' stat.c \
| sort > sym_stat
diff -u sym_stat sym_libc
*/
#endif
/* Also compare with the list in "man 2 statfs" using the
fs-magic-compare make target. */

/* IMPORTANT NOTE: Each of the following 'case S_MAGIC_...:' statements must be followed by a hexadecimal constant in a comment. The S_MAGIC_... name and constant are automatically combined to produce the #define directives in fs.h. */

case S_MAGIC_AAFS: /* 0x5A3C69F0 local */
return "aafs";
case S_MAGIC_ACFS: /* 0x61636673 remote */
return "acfs";
case S_MAGIC_ADFS: /* 0xADF5 local */
return "adfs";
case S_MAGIC_AFFS: /* 0xADFF local */
return "affs";
case S_MAGIC_AFS: /* 0x5346414F remote */
return "afs";
case S_MAGIC_ANON_INODE_FS: /* 0x09041934 local */
return "anon-inode FS";
case S_MAGIC_AUFS: /* 0x61756673 remote */
/* FIXME: change syntax or add an optional attribute like "inotify:no".
The above is labeled as "remote" so that tail always uses polling,
but this isn't really a remote file system type. */
return "aufs";
}

```

```
case S_MAGIC_AUTOFS: /* 0x0187 local */
return "autofs";
case S_MAGIC_BALLOON_KVM: /* 0x13661366 local */
return "balloon-kvm-fs";
case S_MAGIC_BEFS: /* 0x42465331 local */
return "befs";
case S_MAGIC_BDEVFS: /* 0x62646576 local */
return "bdevfs";
case S_MAGIC_BFS: /* 0x1BADFACE local */
return "bfs";
case S_MAGIC_BINDERFS: /* 0x6C6F6F70 local */
return "binderfs";
case S_MAGIC_BPF_FS: /* 0xCAFE4A11 local */
return "bpf_fs";
case S_MAGIC_BINfmtFS: /* 0x42494E4D local */
return "binfmt_misc";
case S_MAGIC_BTRFS: /* 0x9123683E local */
return "btrfs";
case S_MAGIC_BTRFS_TEST: /* 0x73727279 local */
return "btrfs_test";
case S_MAGIC_CEPH: /* 0x00C36400 remote */
return "ceph";
case S_MAGIC_CGROUP: /* 0x0027E0EB local */
return "cgroupfs";
case S_MAGIC_CGROUP2: /* 0x63677270 local */
return "cgroup2fs";
case S_MAGIC_CIFS: /* 0xFF534D42 remote */
return "cifs";
case S_MAGIC_CODA: /* 0x73757245 remote */
return "coda";
case S_MAGIC_COH: /* 0x012FF7B7 local */
return "coh";
case S_MAGIC_CONFIGFS: /* 0x62656570 local */
return "configfs";
case S_MAGIC_CRAMFS: /* 0x28CD3D45 local */
return "cramfs";
case S_MAGIC_CRAMFS_WEND: /* 0x453DCD28 local */
return "cramfs-wend";
case S_MAGIC_DAXFS: /* 0x64646178 local */
return "daxfs";
case S_MAGIC_DEBUGFS: /* 0x64626720 local */
return "debugfs";
case S_MAGIC_DEVFS: /* 0x1373 local */
return "devfs";
case S_MAGIC_DEVMEM: /* 0x454D444D local */
return "devmem";
case S_MAGIC_DEVPTS: /* 0x1CD1 local */
return "devpts";
case S_MAGIC_DMA_BUF: /* 0x444D4142 local */
return "dma-buf-fs";
case S_MAGIC_ECRYPTFS: /* 0xF15F local */
return "ecryptfs";
case S_MAGIC_EFIVARFS: /* 0xDE5E81E4 local */
return "efivars";
case S_MAGIC_EFS: /* 0x00414A53 local */
return "efs";
case S_MAGIC_EROFS_V1: /* 0xE0F5E1E2 local */
return "erofs";
case S_MAGIC_EXFAT: /* 0x2011BAB0 local */
return "exfat";
```

```
case S_MAGIC_EXFS: /* 0x45584653 local */
return "exfs";
case S_MAGIC_EXOFS: /* 0x5DF5 local */
return "exofs";
case S_MAGIC_EXT: /* 0x137D local */
return "ext";
case S_MAGIC_EXT2: /* 0xEF53 local */
return "ext2/ext3";
case S_MAGIC_EXT2_OLD: /* 0xEF51 local */
return "ext2";
case S_MAGIC_F2FS: /* 0xF2F52010 local */
return "f2fs";
case S_MAGIC_FAT: /* 0x4006 local */
return "fat";
case S_MAGIC_FHGFS: /* 0x19830326 remote */
return "fhgfs";
case S_MAGIC_FUSEBLK: /* 0x65735546 remote */
return "fuseblk";
case S_MAGIC_FUSECTL: /* 0x65735543 remote */
return "fusectl";
case S_MAGIC_FUTEXFS: /* 0x0BAD1DEA local */
return "futexfs";
case S_MAGIC_GFS: /* 0x01161970 remote */
return "gfs/gfs2";
case S_MAGIC_GPFS: /* 0x47504653 remote */
return "gpfs";
case S_MAGIC_HFS: /* 0x4244 local */
return "hfs";
case S_MAGIC_HFS_PLUS: /* 0x482B local */
return "hfs+";
case S_MAGIC_HFS_X: /* 0x4858 local */
return "hfsx";
case S_MAGIC_HOSTFS: /* 0x00C0FFEE local */
return "hostfs";
case S_MAGIC_HPFS: /* 0xF995E849 local */
return "hpfs";
case S_MAGIC_HUGETLBFS: /* 0x958458F6 local */
return "hugetlbfs";
case S_MAGIC_MTD_INODE_FS: /* 0x11307854 local */
return "inodes";
case S_MAGIC_IBRIX: /* 0x013111A8 remote */
return "ibrix";
case S_MAGIC_INOTIFYFS: /* 0x2BAD1DEA local */
return "inotifyfs";
case S_MAGIC_ISOFS: /* 0x9660 local */
return "isofs";
case S_MAGIC_ISOFS_R_WIN: /* 0x4004 local */
return "isofs";
case S_MAGIC_ISOFS_WIN: /* 0x4000 local */
return "isofs";
case S_MAGIC_JFFS: /* 0x07C0 local */
return "jffs";
case S_MAGIC_JFFS2: /* 0x72B6 local */
return "jffs2";
case S_MAGIC_JFS: /* 0x3153464A local */
return "jfs";
case S_MAGIC_KAFS: /* 0x6B414653 remote */
return "k-afs";
case S_MAGIC_LOGFS: /* 0xC97E8168 local */
return "logfs";
```

```

case S_MAGIC_LUSTRE: /* 0x0BD00BD0 remote */
return "lustre";
case S_MAGIC_M1FS: /* 0x5346314D local */
return "m1fs";
case S_MAGIC_MINIX: /* 0x137F local */
return "minix";
case S_MAGIC_MINIX_30: /* 0x138F local */
return "minix (30 char.)";
case S_MAGIC_MINIX_V2: /* 0x2468 local */
return "minix v2";
case S_MAGIC_MINIX_V2_30: /* 0x2478 local */
return "minix v2 (30 char.)";
case S_MAGIC_MINIX_V3: /* 0x4D5A local */
return "minix3";
case S_MAGIC_MQUEUE: /* 0x19800202 local */
return "mqueue";
case S_MAGIC_MS DOS: /* 0x4D44 local */
return "msdos";
case S_MAGIC_NCP: /* 0x564C remote */
return "novell";
case S_MAGIC_NFS: /* 0x6969 remote */
return "nfs";
case S_MAGIC_NFSD: /* 0x6E667364 remote */
return "nfsd";
case S_MAGIC NILFS: /* 0x3434 local */
return "nilfs";
case S_MAGIC_NSFS: /* 0x6E736673 local */
return "nsfs";
case S_MAGIC_NTFS: /* 0x5346544E local */
return "ntfs";
case S_MAGIC_OPENPROM: /* 0x9FA1 local */
return "openprom";
case S_MAGIC_OCFS2: /* 0x7461636F remote */
return "ocfs2";
case S_MAGIC_OVERLAYFS: /* 0x794C7630 remote */
/* This may overlay remote file systems.
Also there have been issues reported with inotify and overlayfs,
so mark as "remote" so that polling is used. */
return "overlayfs";
case S_MAGIC_PANFS: /* 0xAAD7AAEA remote */
return "panfs";
case S_MAGIC_PIPEFS: /* 0x50495045 remote */
/* FIXME: change syntax or add an optional attribute like "inotify:no".
pipefs and prlfs are labeled as "remote" so that tail always polls,
but these aren't really remote file system types. */
return "pipefs";
case S_MAGIC_PPC_CMM: /* 0xC7571590 local */
return "ppc-cmm-fs";
case S_MAGIC_PRL_FS: /* 0x7C7C6673 remote */
return "prl_fs";
case S_MAGIC_PROC: /* 0x9FA0 local */
return "proc";
case S_MAGIC_PSTOREFS: /* 0x6165676C local */
return "pstorefs";
case S_MAGIC_QNX4: /* 0x002F local */
return "qnx4";
case S_MAGIC_QNX6: /* 0x68191122 local */
return "qnx6";
case S_MAGIC_RAMFS: /* 0x858458F6 local */
return "ramfs";

```

```

case S_MAGIC_RDTGROUP: /* 0x07655821 local */
return "rdt";
case S_MAGIC_REISERFS: /* 0x52654973 local */
return "reiserfs";
case S_MAGIC_ROMFS: /* 0x7275 local */
return "romfs";
case S_MAGIC_RPC_PIPEFS: /* 0x67596969 local */
return "rpc_pipefs";
case S_MAGIC_SD CARD FS: /* 0x5DCA2DF5 local */
return "sdcardfs";
case S_MAGIC_SECRETMEM: /* 0x5345434D local */
return "secretmem";
case S_MAGIC_SECURITYFS: /* 0x73636673 local */
return "securityfs";
case S_MAGIC_SELINUX: /* 0xF97CFF8C local */
return "selinux";
case S_MAGIC_SMACK: /* 0x43415D53 local */
return "smackfs";
case S_MAGIC_SMB: /* 0x517B remote */
return "smb";
case S_MAGIC_SMB2: /* 0xFE534D42 remote */
return "smb2";
case S_MAGIC_SNFS: /* 0xBEEFDEAD remote */
return "snfs";
case S_MAGIC SOCKFS: /* 0x534F434B local */
return "sockfs";
case S_MAGIC_SQUASHFS: /* 0x73717368 local */
return "squashfs";
case S_MAGIC_SYSFS: /* 0x62656572 local */
return "sysfs";
case S_MAGIC_SYSV2: /* 0x012FF7B6 local */
return "sysv2";
case S_MAGIC_SYSV4: /* 0x012FF7B5 local */
return "sysv4";
case S_MAGIC_TMPFS: /* 0x01021994 local */
return "tmpfs";
case S_MAGIC_TRACEFS: /* 0x74726163 local */
return "tracefs";
case S_MAGIC_UBIFS: /* 0x24051905 local */
return "ubifs";
case S_MAGIC_UDF: /* 0x15013346 local */
return "udf";
case S_MAGIC_UFS: /* 0x00011954 local */
return "ufs";
case S_MAGIC_UFS_BYTESWAPPED: /* 0x54190100 local */
return "ufs";
case S_MAGIC_USBDEVFS: /* 0x9FA2 local */
return "usbdevfs";
case S_MAGIC_V9FS: /* 0x01021997 local */
return "v9fs";
case S_MAGIC_VBOXSF: /* 0x786F4256 remote */
return "vboxsf";
case S_MAGIC_VMHGFS: /* 0xBACBACBC remote */
return "vmhgfs";
case S_MAGIC_VXFS: /* 0xA501FCF5 remote */
/* Veritas File System can run in single instance or clustered mode,
   so mark as remote to cater for the latter case. */
return "vxfs";
case S_MAGIC_VZFS: /* 0x565A4653 local */
return "vzfs";

```

```
case S_MAGIC_WSLFS: /* 0x53464846 local */
return "wslfs";
case S_MAGIC_XENFS: /* 0xABBA1974 local */
return "xenfs";
case S_MAGIC_XENIX: /* 0x012FF7B4 local */
return "xenix";
case S_MAGIC_XFS: /* 0x58465342 local */
return "xfs";
case S_MAGIC_XIAFS: /* 0x012FD16D local */
return "xia";
case S_MAGIC_Z3FOLD: /* 0x0033 local */
return "z3fold";
case S_MAGIC_ZFS: /* 0x2FC12FC1 local */
return "zfs";
case S_MAGIC_ZONEFS: /* 0x5A4F4653 local */
return "zonefs";
case S_MAGIC_ZSMALLOC: /* 0x58295829 local */
return "zsmallocfs";

# elif __GNU__
case FSTYPE_UFS:
return "ufs";
case FSTYPE_NFS:
return "nfs";
case FSTYPE_GFS:
return "gfs";
case FSTYPE_LFS:
return "lfs";
case FSTYPE_SYSV:
return "sysv";
case FSTYPE_FTP:
return "ftp";
case FSTYPE_TAR:
return "tar";
case FSTYPE_AR:
return "ar";
case FSTYPE_CPIO:
return "cpio";
case FSTYPE_MSLOSS:
return "msloss";
case FSTYPE_CPM:
return "cpm";
case FSTYPE_HFS:
return "hfs";
case FSTYPE_DTFS:
return "dtfs";
case FSTYPE_GRFS:
return "grfs";
case FSTYPE_TERM:
return "term";
case FSTYPE_DEV:
return "dev";
case FSTYPE_PROC:
return "proc";
case FSTYPE_IFSOCK:
return "ifsock";
case FSTYPE_AFS:
return "afs";
case FSTYPE_DFS:
```

```

return "dfs";
case FSTYPE_PROC9:
return "proc9";
case FSTYPE_SOCKET:
return "socket";
case FSTYPE_MISC:
return "misc";
case FSTYPE_EXT2FS:
return "ext2/ext3";
case FSTYPE_HTTP:
return "http";
case FSTYPE_MEMFS:
return "memfs";
case FSTYPE_ISO9660:
return "iso9660";
#endif
default:
{
    unsigned long int type = statfsbuf->f_type;
    static char buf[sizeof "UNKNOWN (0x%lx)" - 3
                    + (sizeof type * CHAR_BIT + 3) / 4];
    sprintf (buf, "UNKNOWN (0x%lx)", type);
    return buf;
}
#endif
}

NODISCARD
static char *
human_access (struct stat const *statbuf)
{
    static char modebuf[12];
    filemodestring (statbuf, modebuf);
    modebuf[10] = 0;
    return modebuf;
}

NODISCARD
static char *
human_time (struct timespec t)
{
/* STR must be at least INT_BUFSIZE_BOUND (intmax_t) big, either
   because localtime_rz fails, or because the time zone is truly
   outlandish so that %z expands to a long string. */
    static char str[INT_BUFSIZE_BOUND (intmax_t)
                  + INT_STRLEN_BOUND (int) /* YYYY */
                  + 1 /* because YYYY might equal INT_MAX + 1900 */
                  + sizeof "-MM-DD HH:MM:SS.NNNNNNNNN +"];
    static timezone_t tz;
    if (!tz)
        tz = tzalloc (getenv ("TZ"));
    struct tm tm;
    int ns = t.tv_nsec;
    if (localtime_rz (tz, &t.tv_sec, &tm))
        strftime (str, sizeof str, "%Y-%m-%d %H:%M:%S.%N %z", &tm, tz, ns);
    else
    {
        char sedbuf[INT_BUFSIZE_BOUND (intmax_t)];
        sprintf (str, "%s.%09d", timetostr (t.tv_sec, sedbuf), ns);
    }
}

```

```

        }
    return str;
}

/* PFORMAT points to a '%' followed by a prefix of a format, all of
size PREFIX_LEN. The flags allowed for this format are
ALLOWED_FLAGS; remove other printf flags from the prefix, then
append SUFFIX. */
static void
make_format (char *pformat, size_t prefix_len, char const *allowed_flags,
            char const *suffix)
{
    char *dst = pformat + 1;
    char const *src;
    char const *srclim = pformat + prefix_len;
    for (src = dst; src < srclim && strchr (printf_flags, *src); src++)
        if (strchr (allowed_flags, *src))
            *dst++ = *src;
    while (src < srclim)
        *dst++ = *src++;
    strcpy (dst, suffix);
}

static void
out_string (char *pformat, size_t prefix_len, char const *arg)
{
    make_format (pformat, prefix_len, "-", "s");
    printf (pformat, arg);
}
static int
out_int (char *pformat, size_t prefix_len, intmax_t arg)
{
    make_format (pformat, prefix_len, "'-+ 0", "jd");
    return printf (pformat, arg);
}
static int
out_uint (char *pformat, size_t prefix_len, uintmax_t arg)
{
    make_format (pformat, prefix_len, "'-0", "ju");
    return printf (pformat, arg);
}
static void
out_uint_o (char *pformat, size_t prefix_len, uintmax_t arg)
{
    make_format (pformat, prefix_len, "-#0", "jo");
    printf (pformat, arg);
}
static void
out_uint_x (char *pformat, size_t prefix_len, uintmax_t arg)
{
    make_format (pformat, prefix_len, "-#0", "jx");
    printf (pformat, arg);
}
static int
out_minus_zero (char *pformat, size_t prefix_len)
{
    make_format (pformat, prefix_len, "'-+ 0", ".0f");
    return printf (pformat, -0.25);
}

```

```

/* Output the number of seconds since the Epoch, using a format that
acts like printf's %f format. */
static void
out_epoch_sec (char *pformat, size_t prefix_len,
               struct timespec arg)
{
    char *dot = memchr (pformat, '.', prefix_len);
    size_t sec_prefix_len = prefix_len;
    int width = 0;
    int precision = 0;
    bool frac_left_adjust = false;

    if (dot)
    {
        sec_prefix_len = dot - pformat;
        pformat[prefix_len] = '\0';

        if (ISDIGIT (dot[1]))
        {
            long int lprec = strtol (dot + 1, nullptr, 10);
            precision = (lprec <= INT_MAX ? lprec : INT_MAX);
        }
        else
        {
            precision = 9;
        }

        if (precision && ISDIGIT (dot[-1]))
        {
            /* If a nontrivial width is given, subtract the width of the
               decimal point and PRECISION digits that will be output
               later. */
            char *p = dot;
            *dot = '\0';

            do
                --p;
            while (ISDIGIT (p[-1]));

            long int lwidth = strtol (p, nullptr, 10);
            width = (lwidth <= INT_MAX ? lwidth : INT_MAX);
            if (1 < width)
            {
                p += (*p == '0');
                sec_prefix_len = p - pformat;
                int w_d = (decimal_point_len < width
                           ? width - decimal_point_len
                           : 0);
                if (1 < w_d)
                {
                    int w = w_d - precision;
                    if (1 < w)
                    {
                        char *dst = pformat;
                        for (char const *src = dst; src < p; src++)
                        {
                            if (*src == '-')
                                frac_left_adjust = true;
                            else
                                *dst++ = *src;
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
        sec_prefix_len =
            (dst - pformat
             + (frac_left_adjust ? 0 : sprintf (dst, "%d", w)));
    }
}
}

int divisor = 1;
for (int i = precision; i < 9; i++)
    divisor *= 10;
int frac_sec = arg.tv_nsec / divisor;
int int_len;

if (TYPE_SIGNED (time_t))
{
    bool minus_zero = false;
    if (arg.tv_sec < 0 && arg.tv_nsec != 0)
    {
        int frac_sec_modulus = 1000000000 / divisor;
        frac_sec = (frac_sec_modulus - frac_sec
                    - (arg.tv_nsec % divisor != 0));
        arg.tv_sec += (frac_sec != 0);
        minus_zero = (arg.tv_sec == 0);
    }
    int_len = (minus_zero
               ? out_minus_zero (pformat, sec_prefix_len)
               : out_int (pformat, sec_prefix_len, arg.tv_sec));
}
else
    int_len = out_uint (pformat, sec_prefix_len, arg.tv_sec);

if (precision)
{
    int prec = (precision < 9 ? precision : 9);
    int trailing_prec = precision - prec;
    int ilen = (int_len < 0 ? 0 : int_len);
    int trailing_width = (ilen < width && decimal_point_len < width - ilen
                          ? width - ilen - decimal_point_len - prec
                          : 0);
    printf ("%s%.*d%-*.*d", decimal_point, prec, frac_sec,
            trailing_width, trailing_prec, 0);
}
}

/* Print the context information of FILENAME, and return true iff the
context could not be obtained. */
NODISCARD
static bool
out_file_context (char *pformat, size_t prefix_len, char const *filename)
{
char *scontext;
bool fail = false;

if ((follow_links
     ? getfilecon (filename, &scontext)
     : lgetfilecon (filename, &scontext)) < 0)
{

```

```

error (0, errno, _("failed to get security context of %s"),
       quoteaf (filename));
scontext = nullptr;
fail = true;
}
strcpy (pformat + prefix_len, "s");
printf (pformat, (scontext ? scontext : "?"));
if (scontext)
    freecon (scontext);
return fail;
}

/* Print statfs info. Return zero upon success, nonzero upon failure. */
NODISCARD
static bool
print_statfs (char *pformat, size_t prefix_len, MAYBE_UNUSED char mod, char m,
              int fd, char const *filename,
              void const *data)
{
STRUCT_STATVFS const *statfsbuf = data;
bool fail = false;

switch (m)
{
case 'n':
out_string (pformat, prefix_len, filename);
break;

case 'i':
{
#if STRUCT_STATXFS_F_FSID_IS_INTEGER
    uintmax_t fsid = statfsbuf->f_fsid;
#else
    typedef unsigned int fsid_word;
    static_assert (alignof (STRUCT_STATVFS) % alignof (fsid_word) == 0);
    static_assert (offsetof (STRUCT_STATVFS, f_fsid) % alignof (fsid_word)
                  == 0);
    static_assert (sizeof statfsbuf->f_fsid % alignof (fsid_word) == 0);
    fsid_word const *p = (fsid_word *) &statfsbuf->f_fsid;
#endif
    /* Assume a little-endian word order, as that is compatible
     * with glibc's statvfs implementation. */
    uintmax_t fsid = 0;
    int words = sizeof statfsbuf->f_fsid / sizeof *p;
    for (int i = 0; i < words && i * sizeof *p < sizeof fsid; i++)
    {
        uintmax_t u = p[words - 1 - i];
        fsid |= u << (i * CHAR_BIT * sizeof *p);
    }
#endif
    out_uint_x (pformat, prefix_len, fsid);
}
break;

case 'l':
OUT_NAMEMAX (pformat, prefix_len, SB_F_NAMEMAX (statfsbuf));
break;
case 't':
#if HAVE_STRUCT_STATXFS_F_TYPE
    out_uint_x (pformat, prefix_len, statfsbuf->f_type);

```

```

#else
    fputc ('?', stdout);
#endif
    break;
    case 'T':
        out_string (pformat, prefix_len, human_fstype (statfsbuf));
        break;
    case 'b':
        out_int (pformat, prefix_len, statfsbuf->f_blocks);
        break;
    case 'f':
        out_int (pformat, prefix_len, statfsbuf->f_bfree);
        break;
    case 'a':
        out_int (pformat, prefix_len, statfsbuf->f_bavail);
        break;
    case 's':
        out_uint (pformat, prefix_len, statfsbuf->f_bsize);
        break;
    case 'S':
    {
        uintmax_t frsize = STATFS_FRSIZE (statfsbuf);
        if (!frsize)
            frsize = statfsbuf->f_bszie;
        out_uint (pformat, prefix_len, frsize);
    }
    break;
    case 'c':
        out_uint (pformat, prefix_len, statfsbuf->f_files);
        break;
    case 'd':
        out_int (pformat, prefix_len, statfsbuf->f_ffree);
        break;
    default:
        fputc ('?', stdout);
        break;
    }
return fail;
}

/* Return any bind mounted source for a path.
The caller should not free the returned buffer.
Return nullptr if no bind mount found. */
NODISCARD
static char const *
find_bind_mount (char const * name)
{
    char const * bind_mount = nullptr;

    static struct mount_entry *mount_list;
    static bool tried_mount_list = false;
    if (!tried_mount_list) /* attempt/warn once per process. */
    {
        if (!(mount_list = read_file_system_list (false)))
            error (0, errno, "%s", _("cannot read table of mounted file systems"));
        tried_mount_list = true;
    }

    struct stat name_stats;
    if (stat (name, &name_stats) != 0)

```

```

    return nullptr;

    struct mount_entry *me;
    for (me = mount_list; me; me = me->me_next)
    {
        if (me->me_dummy && me->me_devname[0] == '/'
            && STREQ (me->me_mountdir, name))
        {
            struct stat dev_stats;

            if (stat (me->me_devname, &dev_stats) == 0
                && psame_inode (&name_stats, &dev_stats))
            {
                bind_mount = me->me_devname;
                break;
            }
        }
    }

    return bind_mount;
}

/* Print mount point. Return zero upon success, nonzero upon failure. */
NODISCARD
static bool
out_mount_point (char const *filename, char *pformat, size_t prefix_len,
                 const struct stat *statp)
{
    char const *np = "?", *bp = nullptr;
    char *mp = nullptr;
    bool fail = true;

    /* Look for bind mounts first. Note we output the immediate alias,
       rather than further resolving to a base device mount point. */
    if (follow_links || !S_ISLNK (statp->st_mode))
    {
        char *resolved = canonicalize_file_name (filename);
        if (!resolved)
        {
            error (0, errno, _("failed to canonicalize %s"), quoteaf (filename));
            goto print_mount_point;
        }
        bp = find_bind_mount (resolved);
        free (resolved);
        if (bp)
        {
            fail = false;
            goto print_mount_point;
        }
    }

    /* If there is no direct bind mount, then navigate
       back up the tree looking for a device change.
       Note we don't detect if any of the directory components
       are bind mounted to the same device, but that's OK
       since we've not directly queried them. */
    if ((mp = find_mount_point (filename, statp)))
    {
        /* This dir might be bind mounted to another device,

```

```

    so we resolve the bound source in that case also. */
bp = find_bind_mount (mp);
fail = false;
}

print_mount_point:

out_string (pformat, prefix_len, bp ? bp : mp ? mp : np);
free (mp);
return fail;
}

/* Map a TS with negative TS.tv_nsec to {0,0}. */
static inline struct timespec
neg_to_zero (struct timespec ts)
{
if (0 <= ts.tv_nsec)
    return ts;
struct timespec z = {0};
return z;
}

/* Set the quoting style default if the environment variable
QUOTING_STYLE is set. */

static void
getenv_quoting_style (void)
{
char const *q_style = getenv ("QUOTING_STYLE");
if (q_style)
{
    int i = ARGMATCH (q_style, quoting_style_args, quoting_style_vals);
    if (0 <= i)
        set_quoting_style (nullptr, quoting_style_vals[i]);
    else
    {
        set_quoting_style (nullptr, shell_escape_always_quoting_style);
        error (0, 0, _("ignoring invalid value of environment "
                      "variable QUOTING_STYLE: %s"), quote (q_style));
    }
}
else
    set_quoting_style (nullptr, shell_escape_always_quoting_style);
}

/* Equivalent to quotearg(), but explicit to avoid syntax checks. */
#define quoteN(x) quotearg_style (get_quoting_style (nullptr), x)

/* Output a single-character \ escape. */

static void
print_esc_char (char c)
{
switch (c)
{
case 'a':                  /* Alert. */
    c = '\a';
    break;
case 'b':                  /* Backspace. */
    c = '\b';
}
}
```

```

break;
case 'e':           /* Escape. */
c = '\x1B';
break;
case 'f':           /* Form feed. */
c = '\f';
break;
case 'n':           /* New line. */
c = '\n';
break;
case 'r':           /* Carriage return. */
c = '\r';
break;
case 't':           /* Horizontal tab. */
c = '\t';
break;
case 'v':           /* Vertical tab. */
c = '\v';
break;
case '\"':
case '\\':
break;
default:
error (0, 0, _("warning: unrecognized escape '\\%c'"), c);
break;
}
putchar (c);
}

ATTRIBUTE_PURE
static size_t
format_code_offset (char const *directive)
{
size_t len = strspn (directive + 1, printf_flags);
char const *fmt_char = directive + len + 1;
fmt_char += strspn (fmt_char, digits);
if (*fmt_char == '.')
    fmt_char += 1 + strspn (fmt_char + 1, digits);
return fmt_char - directive;
}

/* Print the information specified by the format string, FORMAT,
calling PRINT_FUNC for each %-directive encountered.
Return zero upon success, nonzero upon failure. */
NODISCARD
static bool
print_it (char const *format, int fd, char const *filename,
    bool (*print_func) (char *, size_t, char, char,
                       int, char const *, void const *),
    void const *data)
{
bool fail = false;

/* Add 2 to accommodate our conversion of the stat '%s' format string
to the longer printf '%llu' one. */
enum
{
MAX_ADDITIONAL_BYTES =
(MAX (sizeof "jd",
      MAX (sizeof "jo", MAX (sizeof "ju", sizeof "jx"))))

```

```

        - 1)
    };
size_t n_alloc = strlen (format) + MAX_ADDITIONAL_BYTES + 1;
char *dest = xmalloc (n_alloc);
char const *b;
for (b = format; *b; b++)
{
switch (*b)
{
case '%':
{
    size_t len = format_code_offset (b);
    char fmt_char = *(b + len);
    char mod_char = 0;
    memcpy (dest, b, len);
    b += len;

    switch (fmt_char)
    {
    case '\0':
        --b;
        FALLTHROUGH;
    case '%':
        if (1 < len)
        {
            dest[len] = fmt_char;
            dest[len + 1] = '\0';
            error (EXIT_FAILURE, 0, _("'%s: invalid directive"),
                  quote (dest));
        }
        putchar ('%');
        break;
    case 'H':
    case 'L':
        mod_char = fmt_char;
        fmt_char = *(b + 1);
        if (print_func == print_stat
            && (fmt_char == 'd' || fmt_char == 'r'))
        {
            b++;
        }
        else
        {
            fmt_char = mod_char;
            mod_char = 0;
        }
        FALLTHROUGH;
    default:
        fail |= print_func (dest, len, mod_char, fmt_char,
                           fd, filename, data);
        break;
    }
    break;
}

case '\\':
if (!interpret_backslash_escapes)
{
    putchar ('\\');
    break;
}

```

```

        }
        ++b;
    if (isodigit (*b))
    {
        int esc_value = octtobin (*b);
        int esc_length = 1; /* number of octal digits */
        for (++b; esc_length < 3 && isodigit (*b);
            ++esc_length, ++b)
        {
            esc_value = esc_value * 8 + octtobin (*b);
        }
        putchar (esc_value);
        --b;
    }
    else if (*b == 'x' && c_isxdigit (to_uchar (b[1])))
    {
        int esc_value = hextobin (b[1]); /* Value of \xhh escape. */
        /* A hexadecimal \xhh escape sequence must have
           1 or 2 hex. digits. */
        ++b;
        if (c_isxdigit (to_uchar (b[1])))
        {
            ++b;
            esc_value = esc_value * 16 + hextobin (*b);
        }
        putchar (esc_value);
    }
    else if (*b == '\0')
    {
        error (0, 0, _("warning: backslash at end of format"));
        putchar ('\\');
        /* Arrange to exit the loop. */
        --b;
    }
    else
    {
        print_esc_char (*b);
    }
    break;

    default:
    putchar (*b);
    break;
}
}
free (dest);

fputs (trailing_delim, stdout);

return fail;
}

/* Stat the file system and print what we find. */
NODISCARD
static bool
do_statfs (char const *filename, char const *format)
{
STRUCT_STATVFS statfsbuf;

if (STREQ (filename, "-"))

```

```

{
error (0, 0, _("using %s to denote standard input does not work"
               " in file system mode"), quoteaf (filename));
return false;
}

if (STATFS (filename, &statfsbuf) != 0)
{
error (0, errno, _("cannot read file system information for %s"),
       quoteaf (filename));
return false;
}

bool fail = print_it (format, -1, filename, print_statfs, &statfsbuf);
return ! fail;
}

struct print_args {
struct stat *st;
struct timespec btime;
};

/* Ask statx to avoid syncing? */
static bool dont_sync;

/* Ask statx to force sync? */
static bool force_sync;

#if USE_STATX
static unsigned int
fmt_to_mask (char fmt)
{
switch (fmt)
{
case 'N':
return STATX_MODE;
case 'd':
case 'D':
return STATX_MODE;
case 'i':
return STATX_INO;
case 'a':
case 'A':
return STATX_MODE;
case 'f':
return STATX_MODE|STATX_TYPE;
case 'F':
return STATX_TYPE;
case 'h':
return STATX_NLINK;
case 'u':
case 'U':
return STATX_UID;
case 'g':
case 'G':
return STATX_GID;
case 'm':
return STATX_MODE|STATX_INO;
case 's':
return STATX_SIZE;
}
}

```

```

        case 't':
        case 'T':
            return STATX_MODE;
        case 'b':
            return STATX_BLOCKS;
        case 'w':
        case 'W':
            return STATX_BTIME;
        case 'x':
        case 'X':
            return STATX_ATIME;
        case 'y':
        case 'Y':
            return STATX_MTIME;
        case 'z':
        case 'Z':
            return STATX_CTIME;
    }
    return 0;
}

ATTRIBUTE_PURE
static unsigned int
format_to_mask (char const *format)
{
    unsigned int mask = 0;
    char const *b;

    for (b = format; *b; b++)
    {
        if (*b != '%')
            continue;

        b += format_code_offset (b);
        if (*b == '\0')
            break;
        mask |= fmt_to_mask (*b);
    }
    return mask;
}

/* statx the file and print what we find */
NODISCARD
static bool
do_stat (char const *filename, char const *format, char const *format2)
{
    int fd = STREQ (filename, "-") ? 0 : AT_FDCWD;
    int flags = 0;
    struct stat st;
    struct statx stx = {0};
    char const *pathname = filename;
    struct print_args pa;
    pa.st = &st;
    pa.btime = (struct timespec) { .tv_sec = -1, .tv_nsec = -1 };

    if (AT_FDCWD != fd)
    {
        pathname = "";
        flags = AT_EMPTY_PATH;
    }

```

```

else if (!follow_links)
{
    flags = AT_SYMLINK_NOFOLLOW;
}

if (dont_sync)
    flags |= AT_STATX_DONT_SYNC;
else if (force_sync)
    flags |= AT_STATX_FORCE_SYNC;

if (! force_sync)
    flags |= AT_NO_AUTOMOUNT;

fd = statx (fd, pathname, flags, format_to_mask (format), &stx);
if (fd < 0)
{
    if (flags & AT_EMPTY_PATH)
        error (0, errno, _("cannot stat standard input"));
    else
        error (0, errno, _("cannot statx %s"), quoteaf (filename));
    return false;
}

if (S_ISBLK (stx.stx_mode) || S_ISCHR (stx.stx_mode))
    format = format2;

statx_to_stat (&stx, &st);
if (stx.stx_mask & STATX_BTIME)
    pa.btime = statx_timestamp_to_timespec (stx.stx_btime);

bool fail = print_it (format, fd, filename, print_stat, &pa);
return ! fail;
}

#ifndef USE_STATX
static struct timespec
get_birthtime (int fd, char const *filename, struct stat const *st)
{
    struct timespec ts = get_stat_birthtime (st);

    #if HAVE_GETATTRAT
    if (ts.tv_nsec < 0)
    {
        nvlist_t *response;
        if ((fd < 0)
            ? setattrat (AT_FDCWD, XATTR_VIEW_READWRITE, filename, &response)
            : fgetattr (fd, XATTR_VIEW_READWRITE, &response))
            == 0)
        {
            uint64_t *val;
            uint_t n;
            if (nvlist_lookup_uint64_array (response, A_CRTIME, &val, &n) == 0
                && 2 <= n
                && val[0] <= TYPE_MAXIMUM (time_t)
                && val[1] < 1000000000 * 2 /* for leap seconds */)
            {
                ts.tv_sec = val[0];
                ts.tv_nsec = val[1];
            }
        }
    }

```

```

        nvlist_free (response);
    }
}

#endif

return ts;
}

/* stat the file and print what we find */
NODISCARD
static bool
do_stat (char const *filename, char const *format,
         char const *format2)
{
int fd = STREQ (filename, "-") ? 0 : -1;
struct stat statbuf;
struct print_args pa;
pa.st = &statbuf;
pa.btime = (struct timespec) {.tv_sec = -1, .tv_nsec = -1};

if (0 <= fd)
{
    if (fstat (fd, &statbuf) != 0)
    {
        error (0, errno, _("cannot stat standard input"));
        return false;
    }
}
/* We can't use the shorter
   (follow_links?stat:Istat) (filename, &statbug)
   since stat might be a function-like macro. */
else if ((follow_links
          ? stat (filename, &statbuf)
          : Istat (filename, &statbuf)) != 0)
{
    error (0, errno, _("cannot stat %s"), quoteaf (filename));
    return false;
}

if (S_ISBLK (statbuf.st_mode) || S_ISCHR (statbuf.st_mode))
    format = format2;

bool fail = print_it (format, fd, filename, print_stat, &pa);
return ! fail;
}
#endif /* USE_STATX */

/* POSIX requires 'ls' to print file sizes without a sign, even
   when negative. Be consistent with that. */

static uintmax_t
unsigned_file_size (off_t size)
{
return size + (size < 0) * ((uintmax_t) OFF_T_MAX - OFF_T_MIN + 1);
}

/* Print stat info. Return zero upon success, nonzero upon failure. */
static bool
print_stat (char *pformat, size_t prefix_len, char mod, char m,

```

```

    int fd, char const *filename, void const *data)
{
    struct print_args *parg = (struct print_args *) data;
    struct stat *statbuf = parg->st;
    struct timespec btime = parg->btime;
    struct passwd *pw_ent;
    struct group *gw_ent;
    bool fail = false;

    switch (m)
    {
        case 'n':
            out_string (pformat, prefix_len, filename);
            break;
        case 'N':
            out_string (pformat, prefix_len, quoteN (filename));
            if (S_ISLNK (statbuf->st_mode))
            {
                char *linkname = areadlink_with_size (filename, statbuf->st_size);
                if (linkname == nullptr)
                {
                    error (0, errno, _("cannot read symbolic link %s"),
                           quoteaf (filename));
                    return true;
                }
                printf (" -> ");
                out_string (pformat, prefix_len, quoteN (linkname));
                free (linkname);
            }
            break;
        case 'd':
            if (mod == 'H')
                out_uint (pformat, prefix_len, major (statbuf->st_dev));
            else if (mod == 'L')
                out_uint (pformat, prefix_len, minor (statbuf->st_dev));
            else
                out_uint (pformat, prefix_len, statbuf->st_dev);
            break;
        case 'D':
            out_uint_x (pformat, prefix_len, statbuf->st_dev);
            break;
        case 'i':
            out_uint (pformat, prefix_len, statbuf->st_ino);
            break;
        case 'a':
            out_uint_o (pformat, prefix_len, statbuf->st_mode & CHMOD_MODE_BITS);
            break;
        case 'A':
            out_string (pformat, prefix_len, human_access (statbuf));
            break;
        case 'f':
            out_uint_x (pformat, prefix_len, statbuf->st_mode);
            break;
        case 'F':
            out_string (pformat, prefix_len, file_type (statbuf));
            break;
        case 'h':
            out_uint (pformat, prefix_len, statbuf->st_nlink);
            break;
        case 'u':

```

```

out_uint (pformat, prefix_len, statbuf->st_uid);
break;
case 'U':
pw_ent = getpwuid (statbuf->st_uid);
out_string (pformat, prefix_len,
            pw_ent ? pw_ent->pw_name : "UNKNOWN");
break;
case 'g':
out_uint (pformat, prefix_len, statbuf->st_gid);
break;
case 'G':
gw_ent = getgrgid (statbuf->st_gid);
out_string (pformat, prefix_len,
            gw_ent ? gw_ent->gr_name : "UNKNOWN");
break;
case 'm':
fail |= out_mount_point (filename, pformat, prefix_len, statbuf);
break;
case 's':
out_uint (pformat, prefix_len, unsigned_file_size (statbuf->st_size));
break;
case 'r':
if (mod == 'H')
    out_uint (pformat, prefix_len, major (statbuf->st_rdev));
else if (mod == 'L')
    out_uint (pformat, prefix_len, minor (statbuf->st_rdev));
else
    out_uint (pformat, prefix_len, statbuf->st_rdev);
break;
case 'R':
out_uint_x (pformat, prefix_len, statbuf->st_rdev);
break;
case 't':
out_uint_x (pformat, prefix_len, major (statbuf->st_rdev));
break;
case 'T':
out_uint_x (pformat, prefix_len, minor (statbuf->st_rdev));
break;
case 'B':
out_uint (pformat, prefix_len, ST_NBLOCKSIZE);
break;
case 'b':
out_uint (pformat, prefix_len, STP_NBLOCKS (statbuf));
break;
case 'o':
out_uint (pformat, prefix_len, STP_BLKSIZE (statbuf));
break;
case 'w':
{
#endif ! USE_STATX
    btime = get_birthtime (fd, filename, statbuf);
#endif
    if (btime.tv_nsec < 0)
        out_string (pformat, prefix_len, "-");
    else
        out_string (pformat, prefix_len, human_time (btime));
}
break;
case 'W':
{

```

```

#if ! USE_STATX
    btime = get_birthtime (fd, filename, statbuf);
#endif
    out_epoch_sec (pformat, prefix_len, neg_to_zero (btime));
}
break;
case 'x':
out_string (pformat, prefix_len, human_time (get_stat_atime (statbuf)));
break;
case 'X':
out_epoch_sec (pformat, prefix_len, get_stat_atime (statbuf));
break;
case 'y':
out_string (pformat, prefix_len, human_time (get_stat_mtime (statbuf)));
break;
case 'Y':
out_epoch_sec (pformat, prefix_len, get_stat_mtime (statbuf));
break;
case 'z':
out_string (pformat, prefix_len, human_time (get_stat_ctime (statbuf)));
break;
case 'Z':
out_epoch_sec (pformat, prefix_len, get_stat_ctime (statbuf));
break;
case 'C':
fail |= out_file_context (pformat, prefix_len, filename);
break;
default:
fputc ('?', stdout);
break;
}
return fail;
}

/* Return an allocated format string in static storage that
corresponds to whether FS and TERSE options were declared. */
static char *
default_format (bool fs, bool terse, bool device)
{
char *format;
if (fs)
{
if (terse)
format = xstrdup (fmt_terse_fs);
else
{
/* TRANSLATORS: This string uses format specifiers from
'stat --help' with --file-system, and NOT from printf. */
format = xstrdup (_(" File: \"%n\"\n"
" ID: %-8i Namelen: %-7l Type: %T\n"
" Block size: %-10s Fundamental block size: %S\n"
" Blocks: Total: %-10b Free: %-10f Available: %a\n"
" Inodes: Total: %-10c Free: %d\n"));
}
}
else /* ! fs */
{
if (terse)
{
if (0 < is_selinux_enabled ())

```

```

        format = xstrdup (fmt_terse_selinux);
    else
        format = xstrdup (fmt_terse_regular);
    }
else
{
    char *temp;
    /* TRANSLATORS: This string uses format specifiers from
       'stat --help' without --file-system, and NOT from printf. */
    format = xstrdup (_("\
File: %N\n\
Size: %-10s\tBlocks: %-10b IO Block: %-6o %F\n\
"));

    temp = format;
    if (device)
    {
        /* TRANSLATORS: This string uses format specifiers from
           'stat --help' without --file-system, and NOT from printf. */
        format = xasprintf ("%s%s", format, _("\
" "Device: %Hd,%Ld\tInode: %-10i Links: %-5h Device type: %Hr,%Lr\n\
")));
    }
    else
    {
        /* TRANSLATORS: This string uses format specifiers from
           'stat --help' without --file-system, and NOT from printf. */
        format = xasprintf ("%s%s", format, _("\
" "Device: %Hd,%Ld\tInode: %-10i Links: %h\n\
")));
    }
    free (temp);

    temp = format;
    /* TRANSLATORS: This string uses format specifiers from
       'stat --help' without --file-system, and NOT from printf. */
    format = xasprintf ("%s%s", format, _("\
" "Access: (%04a/%10.10A) Uid: (%5u/%8U) Gid: (%5g/%8G)\n\
")));
    free (temp);

    if (0 < is_selinux_enabled ())
    {
        temp = format;
        /* TRANSLATORS: This string uses format specifiers from
           'stat --help' without --file-system, and NOT from printf. */
        format = xasprintf ("%s%s", format, _("Context: %C\n"));
        free (temp);
    }

    temp = format;
    /* TRANSLATORS: This string uses format specifiers from
       'stat --help' without --file-system, and NOT from printf. */
    format = xasprintf ("%s%s", format,
        _("Access: %x\n"
        "Modify: %y\n"
        "Change: %z\n"
        " Birth: %w\n"));
    free (temp);
}

```

```

        }
    return format;
}

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    {
        printf (_("Usage: %s [OPTION]... FILE...\n"), program_name);
        fputs (_("\
Display file or file system status.\n\
"), stdout);

        emit_mandatory_arg_note ();

        fputs (_("\
-L, --dereference      follow links\n\
-f, --file-system      display file system status instead of file status\n\
"), stdout);
        fputs (_("\
--cached=MODE      specify how to use cached attributes;\n\
                  useful on remote file systems. See MODE below\n\
"), stdout);
        fputs (_("\
-c --format=FORMAT   use the specified FORMAT instead of the default;\n\
                  output a newline after each use of FORMAT\n\
--printf=FORMAT     like --format, but interpret backslash escapes,\n\
                  and do not output a mandatory trailing newline;\n\
                  if you want a newline, include \\n in FORMAT\n\
"), stdout);
        fputs (_("\
-t, --terse          print the information in terse form\n\
"), stdout);
        fputs (HELP_OPTION_DESCRIPTION, stdout);
        fputs (VERSION_OPTION_DESCRIPTION, stdout);

        fputs (_("\n\
The MODE argument of --cached can be: always, never, or default.\n\
'always' will use cached attributes if available, while\n\
'never' will try to synchronize with the latest attributes, and\n\
'default' will leave it up to the underlying file system.\n\
"), stdout);

        fputs (_("\n\
The valid format sequences for files (without --file-system):\n\
\n\
%a  permission bits in octal (see '#' and '0' printf flags)\n\
%A  permission bits and file type in human readable form\n\
%b  number of blocks allocated (see %B)\n\
%B  the size in bytes of each block reported by %b\n\
%C  SELinux security context string\n\
"), stdout);
        fputs (_("\
%d  device number in decimal (st_dev)\n\
%D  device number in hex (st_dev)\n\
%Hd major device number in decimal\n\
%Ld minor device number in decimal\n\
%f  raw mode in hex\n\
%F  file type\n\

```

```

%g group ID of owner\n\
%G group name of owner\n\
"), stdout);
    fputs (_("\
%h number of hard links\n\
%i inode number\n\
%m mount point\n\
%n file name\n\
%N quoted file name with dereference if symbolic link\n\
%o optimal I/O transfer size hint\n\
%s total size, in bytes\n\
%r device type in decimal (st_rdev)\n\
%R device type in hex (st_rdev)\n\
%Hr major device type in decimal, for character/block device special files\n\
%Lr minor device type in decimal, for character/block device special files\n\
%tr major device type in hex, for character/block device special files\n\
%Tr minor device type in hex, for character/block device special files\n\
"), stdout);
    fputs (_("\
%u user ID of owner\n\
%U user name of owner\n\
%w time of file birth, human-readable; - if unknown\n\
%W time of file birth, seconds since Epoch; 0 if unknown\n\
%x time of last access, human-readable\n\
%X time of last access, seconds since Epoch\n\
%y time of last data modification, human-readable\n\
%Y time of last data modification, seconds since Epoch\n\
%z time of last status change, human-readable\n\
%Z time of last status change, seconds since Epoch\n\
\n\
"), stdout);

    fputs (_("\
Valid format sequences for file systems:\n\
\n\
%a free blocks available to non-superuser\n\
%b total data blocks in file system\n\
%c total file nodes in file system\n\
%d free file nodes in file system\n\
%f free blocks in file system\n\
"), stdout);
    fputs (_("\
%i file system ID in hex\n\
%l maximum length of filenames\n\
%n file name\n\
%s block size (for faster transfers)\n\
%S fundamental block size (for block counts)\n\
%t file system type in hex\n\
%T file system type in human readable form\n\
"), stdout);

    printf (_("\n\
--terse is equivalent to the following FORMAT:\n\
%s\n\
"),
#if HAVE_SELINUX_SELINUX_H
    fmt_terse_selinux
#else
    fmt_terse_regular
#endif

```

```

    );
    printf (_("\
--terse --file-system is equivalent to the following FORMAT:\n\
% s\
"), fmt_terse_fs);

    printf (USAGE_BUILTIN_WARNING, PROGRAM_NAME);
    emit_ancillary_info (PROGRAM_NAME);
}
exit (status);
}

int
main (int argc, char *argv[])
{
int c;
bool fs = false;
bool terse = false;
char *format = nullptr;
char *format2;
bool ok = true;

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

struct lconv const *locale = localeconv ();
decimal_point = (locale->decimal_point[0] ? locale->decimal_point : ".");
decimal_point_len = strlen (decimal_point);

atexit (close_stdout);

while ((c = getopt_long (argc, argv, "c:fLt", long_options, nullptr)) != -1)
{
switch (c)
{
case PRINTF_OPTION:
format = optarg;
interpret_backslash_escapes = true;
trailing_delim = "";
break;

case 'c':
format = optarg;
interpret_backslash_escapes = false;
trailing_delim = "\n";
break;

case 'L':
follow_links = true;
break;

case 'f':
fs = true;
break;

case 't':

```

```

terse = true;
break;

case 0:
switch (XARGMATCH ("--cached", optarg, cached_args, cached_modes))
{
    case cached_never:
        force_sync = true;
        dont_sync = false;
        break;
    case cached_always:
        force_sync = false;
        dont_sync = true;
        break;
    case cached_default:
        force_sync = false;
        dont_sync = false;
}
break;

case_GETOPT_HELP_CHAR;

case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

default:
usage (EXIT_FAILURE);
}
}

if (argc == optind)
{
error (0, 0, _("missing operand"));
usage (EXIT_FAILURE);
}

if (format)
{
if (strstr (format, "%N"))
    getenv_quoting_style ();
format2 = format;
}
else
{
format = default_format (fs, terse, /* device= */ false);
format2 = default_format (fs, terse, /* device= */ true);
}

for (int i = optind; i < argc; i++)
ok &= (fs
    ? do_statfs (argv[i], format)
    : do_stat (argv[i], format, format2));

main_exit (ok ? EXIT_SUCCESS : EXIT_FAILURE);
}

""",
{
    "error_category": "Missing Struct Members",
    "error": "no member named 'f_fsid' in 'statfs'"
},

```

```
{
  "error_category": "Compile-Time Validation Failures",
  "error": "static_assert (offsetof (STRUCT_STATVFS, f_fsid) % alignof (fsid_word) == 0)"
},
{
  "error_category": "Compile-Time Validation Failures",
  "error": "static_assert (sizeof statfsbuf->f_fsid % alignof (fsid_word) == 0)"
},
{
  "error_category": "Missing Struct Members",
  "error": "fsid_word const *p = (fsid_word *) &statfsbuf->f_fsid"
},
{
  "error_category": "Missing Struct Members",
  "error": "int words = sizeof statfsbuf->f_fsid / sizeof *p"
},
{
  "error_category": "Missing Struct Members",
  "error": "no member named 'f_blocks' in 'struct statfs'"
},
{
  "error_category": "Missing Struct Members",
  "error": "no member named 'f_bfree' in 'struct statfs'"
},
{
  "error_category": "Missing Struct Members",
  "error": "no member named 'f_bavail' in 'struct statfs'"
},
{
  "error_category": "Missing Struct Members",
  "error": "no member named 'f_bsize' in 'struct statfs'"
},
{
  "error_category": "Missing Struct Members",
  "error": "no member named 'f_files' in 'struct statfs'"
},
{
  "error_category": "Missing Struct Members",
  "error": "no member named 'f_ffree' in 'struct statfs'"
},
{
  "error_category": "Function Declaration Issues",
  "error": "implicit declaration of function 'statfs' is invalid in C99 [-Wimplicit-function-declaration]"
}
}
"correct_code":
"""
/* stat.c -- display file or file system status
Copyright (C) 2001-2024 Free Software Foundation, Inc.
```

This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

Written by Michael Meskes. \*/

```
#include <config.h>

/* Keep this conditional in sync with the similar conditional in
..../m4/stat-prog.m4. */
#ifndef (STAT_STATVFS || STAT_STATVFS64)
    \
    && (HAVE_STRUCT_STATVFS_F_BASETYPE ||
HAVE_STRUCT_STATVFS_F_FSTYPENAME \
    || (! HAVE_STRUCT_STATFS_F_FSTYPENAME &&
HAVE_STRUCT_STATVFS_F_TYPE)))
#define USE_STATVFS 1
#else
#define USE_STATVFS 0
#endif

#include <stdio.h>
#include <sys/types.h>
#include <pwd.h>
#include <grp.h>
#if USE_STATVFS
#include <sys/statvfs.h>
#elif HAVE_SYS_VFS_H
#include <sys/vfs.h>
#elif HAVE_SYS_MOUNT_H && HAVE_SYS_PARAM_H
/* NOTE: freebsd5.0 needs sys/param.h and sys/mount.h for statfs.
It does have statvfs.h, but shouldn't use it, since it doesn't
HAVE_STRUCT_STATVFS_F_BASETYPE. So find a clean way to fix it. */
/* NetBSD 1.5.2 needs these, for the declaration of struct statfs. */
#include <sys/param.h>
#include <sys/mount.h>
#if HAVE_NFS_NFS_CLNT_H && HAVE_NFS_VFS_H
/* Ultrix 4.4 needs these for the declaration of struct statfs. */
#include <netinet/in.h>
#include <nfs/nfs_clnt.h>
#include <nfs/vfs.h>
#endif
#elif HAVE_OS_H /* BeOS */
#include <fs_info.h>
#endif
#include <selinux/selinux.h>
#include <getopt.h>

#include "system.h"

#include "areadlink.h"
#include "argmatch.h"
#include "c-ctype.h"
#include "file-type.h"
#include "filemode.h"
#include "fs.h"
#include "mountlist.h"
#include "quote.h"
#include "stat-size.h"
#include "stat-time.h"
#include "strftime.h"
```

```

#include "find-mount-point.h"
#include "xvasprintf.h"
#include "statx.h"

#if HAVE_STATX && defined STATX_INO
# define USE_STATX 1
#else
# define USE_STATX 0
#endif

#if USE_STATVFS
# define STRUCT_STATXFS_F_FSID_IS_INTEGER
STRUCT_STATVFS_F_FSID_IS_INTEGER
# define HAVE_STRUCT_STATXFS_F_TYPE HAVE_STRUCT_STATVFS_F_TYPE
# if HAVE_STRUCT_STATVFS_F_NAMEMAX
# define SB_F_NAMEMAX(S) ((S)->f_namemax)
# endif
# if !STAT_STATVFS && STAT_STATVFS64
# define STRUCT_STATVFS struct statvfs64
# define STATFS statvfs64
# else
# define STRUCT_STATVFS struct statvfs
# define STATFS statvfs
# endif
# define STATFS_FRSIZE(S) ((S)->f_frsize)
#else
# define HAVE_STRUCT_STATXFS_F_TYPE HAVE_STRUCT_STATFS_F_TYPE
# if HAVE_STRUCT_STATFS_F_NAMELEN
# define SB_F_NAMEMAX(S) ((S)->f_namelen)
# elif HAVE_STRUCT_STATFS_F_NAMEMAX
# define SB_F_NAMEMAX(S) ((S)->f_namemax)
# endif
#endif

#ifndef __MVS__
#include <sys/statfs.h>
# define STATFS w_statfs
#else
# define STATFS statfs
#endif

# if HAVE_OS_H /* BeOS */
/* BeOS has a statvfs function, but it does not return sensible values
for f_files, f_ffree and f_favail, and lacks f_type, f_basetype and
f_fstypename. Use 'struct fs_info' instead. */
NODISCARD
static int
statfs (char const *filename, struct fs_info *buf)
{
dev_t device = dev_for_path (filename);
if (device < 0)
{
errno = (device == B_ENTRY_NOT_FOUND ? ENOENT
: device == B_BAD_VALUE ? EINVAL
: device == B_NAME_TOO_LONG ? ENAMETOOLONG
: device == B_NO_MEMORY ? ENOMEM
: device == B_FILE_ERROR ? EIO
: 0);
return -1;
}
/* If successful, buf->dev will be == device. */

```

```

return fs_stat_dev (device, buf);
}
# define f_fsid dev
# define f_blocks total_blocks
# define f_bfree free_blocks
# define f_bavail free_blocks
# define f_bsize io_size
# define f_files total_nodes
# define f_ffree free_nodes
# define STRUCT_STATVFS struct fs_info
# define STRUCT_STATXFS_F_FSID_IS_INTEGER true
# define STATFS_FRSIZE(S) ((S)->block_size)
# else
#ifndef __MVS__
# define STRUCT_STATVFS struct statfs
#else
# define STRUCT_STATVFS struct w_statfs
#endif
# define STRUCT_STATXFS_F_FSID_IS_INTEGER
STRUCT_STATFS_F_FSID_IS_INTEGER
# if HAVE_STRUCT_STATFS_F_FRSIZE
# define STATFS_FRSIZE(S) ((S)->f_frsize)
# else
# define STATFS_FRSIZE(S) 0
# endif
# endif
#endif

#endif SB_F_NAMEMAX
# define OUT_NAMEMAX out_uint
#else
/* Depending on whether statvfs or statfs is used,
neither f_namemax or f_namelen may be available. */
# define SB_F_NAMEMAX(S) "?"
# define OUT_NAMEMAX out_string
#endif

#if HAVE_STRUCT_STATVFS_F_BASETYPE
# define STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME f_basetype
#else
# if HAVE_STRUCT_STATVFS_F_FSTYPENAME ||
HAVE_STRUCT_STATFS_F_FSTYPENAME
# define STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME f_fstypename
# elif HAVE_OS_H /* BeOS */
# define STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME fsh_name
# endif
#endif

#if HAVE_GETATTRATR
# include <attr.h>
# include <sys/nvpair.h>
#endif

/* FIXME: these are used by printf.c, too */
#define isodigit(c) ('0' <= (c) && (c) <= '7')
#define octtobin(c) ((c) - '0')
#define hextobin(c) ((c) >= 'a' && (c) <= 'f' ? (c) - 'a' + 10 : \
(c) >= 'A' && (c) <= 'F' ? (c) - 'A' + 10 : (c) - '0')

static char const digits[] = "0123456789";

```

```

/* Flags that are portable for use in printf, for at least one
conversion specifier; make_format removes non-portable flags as
needed for particular specifiers. The glibc 2.2 extension "l" is
listed here; it is removed by make_format because it has undefined
behavior elsewhere and because it is incompatible with
out_epoch_sec. */
static char const printf_flags[] = "-+ #0l";

/* Formats for the --terse option. */
static char const fmt_terse_fs[] = "%n %i %l %t %s %S %b %f %a %c %d\n";
static char const fmt_terse_regular[] = "%n %s %b %f %u %g %D %i %h %t %T"
                                         " %X %Y %Z %W %o\n";
static char const fmt_terse_selinux[] = "%n %s %b %f %u %g %D %i %h %t %T"
                                         " %X %Y %Z %W %o %C\n";

#define PROGRAM_NAME "stat"

#define AUTHORS proper_name ("Michael Meskes")

enum
{
PRINTF_OPTION = CHAR_MAX + 1
};

enum cached_mode
{
cached_default,
cached_never,
cached_always
};

static char const *const cached_args[] =
{
"default", "never", "always", nullptr
};

static enum cached_mode const cached_modes[] =
{
cached_default, cached_never, cached_always
};

static struct option const long_options[] =
{
{"dereference", no_argument, nullptr, 'L'},
{"file-system", no_argument, nullptr, 'f'},
{"format", required_argument, nullptr, 'c'},
{"printf", required_argument, nullptr, PRINTF_OPTION},
 {"terse", no_argument, nullptr, 't'},
 {"cached", required_argument, nullptr, 0},
 {GETOPT_HELP_OPTION_DECL},
 {GETOPT_VERSION_OPTION_DECL},
 {nullptr, 0, nullptr, 0}
};

/* Whether to follow symbolic links; True for --dereference (-L). */
static bool follow_links;

/* Whether to interpret backslash-escape sequences.
True for --printf=FMT, not for --format=FMT (-c). */

```

```

static bool interpret_backslash_escapes;

/* The trailing delimiter string:
   "" for --printf=FMT, "\n" for --format=FMT (-c). */
static char const *trailing_delim = "";

/* The representation of the decimal point in the current locale. */
static char const *decimal_point;
static size_t decimal_point_len;

static bool
print_stat (char *pformat, size_t prefix_len, char mod, char m,
            int fd, char const *filename, void const *data);

/* Return the type of the specified file system.
Some systems have statvfs.f_basetype[FSTYPSZ] (AIX, HP-UX, and Solaris).
Others have statvfs.f_fstypename[_VFS_NAMELEN] (NetBSD 3.0).
Others have statfs.f_fstypename[MFSNAMELEN] (NetBSD 1.5.2).
Still others have neither and have to get by with f_type (GNU/Linux).
But f_type may only exist in statfs (Cygwin). */
NODISCARD
static char const *
human_fstype (STRUCT_STATVFS const *statfsbuf)
{
#ifndef STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME
#if defined __MVS__
return 0;
#else
return statfsbuf->STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME;
#endif
#else
switch (statfsbuf->f_type)
{
# if defined __linux__ || defined __ANDROID__

/* Compare with what's in libc:
   f=/a/libc/sysdeps/unix/sysv/linux/linux_fsinfo.h
   sed -n '/ADFS_SUPER_MAGIC/,/SYSFS_MAGIC/p' $f \
   | perl -n -e '/#define (.*)_(?:(SUPER_)MAGIC\s+0x(\S+)/' \
   -e 'and print "case S_MAGIC_$1: /* 0x" . uc($2) . " */\n"' \
   | sort > sym_libc
   perl -ne '/^\s+(case S_MAGIC_.*)?: \/* 0x(\S+) \*/' \
   -e 'and do { $v=uc$2; print "$1: /* 0x$v */\n"}' stat.c \
   | sort > sym_stat
   diff -u sym_stat sym_libc
}

/* Also compare with the list in "man 2 statfs" using the
   fs-magic-compare make target. */

/* IMPORTANT NOTE: Each of the following 'case S_MAGIC_...:' statements must be followed by a hexadecimal constant in a comment. The S_MAGIC_... name and constant are automatically combined to produce the #define directives in fs.h. */

case S_MAGIC_AAFS: /* 0x5A3C69F0 local */
return "aafs";
case S_MAGIC_ACFS: /* 0x61636673 remote */
return "acfs";
case S_MAGIC_ADFS: /* 0xAFD5 local */

```

```

return "adfs";
case S_MAGIC_AFFS: /* 0xADFF local */
return "affs";
case S_MAGIC_AFS: /* 0x5346414F remote */
return "afs";
case S_MAGIC_ANON_INODE_FS: /* 0x09041934 local */
return "anon-inode FS";
case S_MAGIC_AUFS: /* 0x61756673 remote */
/* FIXME: change syntax or add an optional attribute like "inotify:no".
   The above is labeled as "remote" so that tail always uses polling,
   but this isn't really a remote file system type. */
return "aufs";
case S_MAGIC_AUTOFS: /* 0x0187 local */
return "autofs";
case S_MAGIC_BALLOON_KVM: /* 0x13661366 local */
return "balloon-kvm-fs";
case S_MAGIC_BEFS: /* 0x42465331 local */
return "befs";
case S_MAGIC_BDEVFS: /* 0x62646576 local */
return "bdevfs";
case S_MAGIC_BFS: /* 0x1BADFACE local */
return "bfs";
case S_MAGIC_BINDERFS: /* 0x6C6F6F70 local */
return "binderfs";
case S_MAGIC_BPF_FS: /* 0xCAFE4A11 local */
return "bpf_fs";
case S_MAGIC_BINfmtFS: /* 0x42494E4D local */
return "binfmt_misc";
case S_MAGIC_BTRFS: /* 0x9123683E local */
return "btrfs";
case S_MAGIC_BTRFS_TEST: /* 0x73727279 local */
return "btrfs_test";
case S_MAGIC_CEPH: /* 0x00C36400 remote */
return "ceph";
case S_MAGIC_CGROUP: /* 0x0027E0EB local */
return "cgroupfs";
case S_MAGIC_CGROUP2: /* 0x63677270 local */
return "cgroup2fs";
case S_MAGIC_CIFS: /* 0xFF534D42 remote */
return "cifs";
case S_MAGIC_CODA: /* 0x73757245 remote */
return "coda";
case S_MAGIC_COH: /* 0x012FF7B7 local */
return "coh";
case S_MAGIC_CONFIGFS: /* 0x62656570 local */
return "configfs";
case S_MAGIC_CRAMFS: /* 0x28CD3D45 local */
return "cramfs";
case S_MAGIC_CRAMFS_WEND: /* 0x453DCD28 local */
return "cramfs-wend";
case S_MAGIC_DAXFS: /* 0x64646178 local */
return "daxfs";
case S_MAGIC_DEBUGFS: /* 0x64626720 local */
return "debugfs";
case S_MAGIC_DEVFS: /* 0x1373 local */
return "devfs";
case S_MAGIC_DEVMEM: /* 0x454D444D local */
return "devmem";
case S_MAGIC_DEVPTS: /* 0x1CD1 local */
return "devpts";

```

```
case S_MAGIC_DMA_BUF: /* 0x444D4142 local */
return "dma-buf-fs";
case S_MAGIC_ECRYPTFS: /* 0xF15F local */
return "ecryptfs";
case S_MAGIC_EFIVARFS: /* 0xDE5E81E4 local */
return "efivarfs";
case S_MAGIC_EFS: /* 0x00414A53 local */
return "efs";
case S_MAGICEROFS_V1: /* 0xE0F5E1E2 local */
return "erofs";
case S_MAGIC_EXFAT: /* 0x2011BAB0 local */
return "exfat";
case S_MAGIC_EXFS: /* 0x45584653 local */
return "exfs";
case S_MAGIC_EXOFS: /* 0x5DF5 local */
return "exofs";
case S_MAGIC_EXT: /* 0x137D local */
return "ext";
case S_MAGIC_EXT2: /* 0xEF53 local */
return "ext2/ext3";
case S_MAGIC_EXT2_OLD: /* 0xEF51 local */
return "ext2";
case S_MAGIC_F2FS: /* 0xF2F52010 local */
return "f2fs";
case S_MAGIC_FAT: /* 0x4006 local */
return "fat";
case S_MAGIC_FHGFS: /* 0x19830326 remote */
return "fhgfs";
case S_MAGIC_FUSEBLK: /* 0x65735546 remote */
return "fuseblk";
case S_MAGIC_FUSECTL: /* 0x65735543 remote */
return "fusectl";
case S_MAGIC_FUTEXFS: /* 0x0BAD1DEA local */
return "futexfs";
case S_MAGIC_GFS: /* 0x01161970 remote */
return "gfs/gfs2";
case S_MAGIC_GPFS: /* 0x47504653 remote */
return "gpfs";
case S_MAGIC_HFS: /* 0x4244 local */
return "hfs";
case S_MAGIC_HFS_PLUS: /* 0x482B local */
return "hfs+";
case S_MAGIC_HFS_X: /* 0x4858 local */
return "hfsx";
case S_MAGIC_HOSTFS: /* 0x00C0FFEE local */
return "hostfs";
case S_MAGIC_HPFS: /* 0xF995E849 local */
return "hpfs";
case S_MAGIC_HUGETLBFS: /* 0x958458F6 local */
return "hugetlbfs";
case S_MAGIC_MTD_INODE_FS: /* 0x11307854 local */
return "inodes";
case S_MAGIC_IBRIX: /* 0x013111A8 remote */
return "ibrix";
case S_MAGIC_INOTIFYFS: /* 0x2BAD1DEA local */
return "inotifyfs";
case S_MAGIC_ISOFS: /* 0x9660 local */
return "isofs";
case S_MAGIC_ISOFS_R_WIN: /* 0x4004 local */
return "isofs";
```

```

case S_MAGIC_ISOFS_WIN: /* 0x4000 local */
return "isofs";
case S_MAGIC_JFFS: /* 0x07C0 local */
return "jffs";
case S_MAGIC_JFFS2: /* 0x72B6 local */
return "jffs2";
case S_MAGIC_JFS: /* 0x3153464A local */
return "jfs";
case S_MAGIC_KAFS: /* 0x6B414653 remote */
return "k-afs";
case S_MAGIC_LOGFS: /* 0xC97E8168 local */
return "logfs";
case S_MAGIC_LUSTRE: /* 0x0BD00BD0 remote */
return "lustre";
case S_MAGIC_M1FS: /* 0x5346314D local */
return "m1fs";
case S_MAGIC_MINIX: /* 0x137F local */
return "minix";
case S_MAGIC_MINIX_30: /* 0x138F local */
return "minix (30 char.)";
case S_MAGIC_MINIX_V2: /* 0x2468 local */
return "minix v2";
case S_MAGIC_MINIX_V2_30: /* 0x2478 local */
return "minix v2 (30 char.)";
case S_MAGIC_MINIX_V3: /* 0x4D5A local */
return "minix3";
case S_MAGIC_MQUEUE: /* 0x19800202 local */
return "mqueue";
case S_MAGIC_MS DOS: /* 0x4D44 local */
return "msdos";
case S_MAGIC_NCP: /* 0x564C remote */
return "novell";
case S_MAGIC_NFS: /* 0x6969 remote */
return "nfs";
case S_MAGIC_NFSD: /* 0x6E667364 remote */
return "nfsd";
case S_MAGIC NILFS: /* 0x3434 local */
return "nilfs";
case S_MAGIC_NSFS: /* 0x6E736673 local */
return "nsfs";
case S_MAGIC_NTFS: /* 0x5346544E local */
return "ntfs";
case S_MAGIC_OPENPROM: /* 0x9FA1 local */
return "openprom";
case S_MAGIC_OCF S2: /* 0x7461636F remote */
return "ocfs2";
case S_MAGIC_OVERLAYFS: /* 0x794C7630 remote */
/* This may overlay remote file systems.
Also there have been issues reported with inotify and overlayfs,
so mark as "remote" so that polling is used. */
return "overlayfs";
case S_MAGIC_PANFS: /* 0xAAD7AAEA remote */
return "panfs";
case S_MAGIC_PIPEFS: /* 0x50495045 remote */
/* FIXME: change syntax or add an optional attribute like "inotify:no".
pipefs and prlfs are labeled as "remote" so that tail always polls,
but these aren't really remote file system types. */
return "pipefs";
case S_MAGIC_PPC_CMM: /* 0xC7571590 local */
return "ppc-cmm-fs";

```

```
case S_MAGIC_PRL_FS: /* 0x7C7C6673 remote */
return "prl_fs";
case S_MAGIC_PROC: /* 0x9FA0 local */
return "proc";
case S_MAGIC_PSTOREFS: /* 0x6165676C local */
return "pstorefs";
case S_MAGIC_QNX4: /* 0x002F local */
return "qnx4";
case S_MAGIC_QNX6: /* 0x68191122 local */
return "qnx6";
case S_MAGIC_RAMFS: /* 0x858458F6 local */
return "ramfs";
case S_MAGIC_RDTGROUP: /* 0x07655821 local */
return "rdt";
case S_MAGIC_REISERFS: /* 0x52654973 local */
return "reiserfs";
case S_MAGIC_ROMFS: /* 0x7275 local */
return "romfs";
case S_MAGIC_RPC_PIPEFS: /* 0x67596969 local */
return "rpc_pipefs";
case S_MAGIC_SD CARD FS: /* 0x5DCA2DF5 local */
return "sdcardfs";
case S_MAGIC_SECRETMEM: /* 0x5345434D local */
return "secretmem";
case S_MAGIC_SECURITYFS: /* 0x73636673 local */
return "securityfs";
case S_MAGIC_SELINUX: /* 0xF97CFF8C local */
return "selinux";
case S_MAGIC_SMACK: /* 0x43415D53 local */
return "smackfs";
case S_MAGIC_SMB: /* 0x517B remote */
return "smb";
case S_MAGIC_SMB2: /* 0xFE534D42 remote */
return "smb2";
case S_MAGIC_SNFS: /* 0xBEEFDEAD remote */
return "snfs";
case S_MAGIC SOCK FS: /* 0x534F434B local */
return "sockfs";
case S_MAGIC_SQUASHFS: /* 0x73717368 local */
return "squashfs";
case S_MAGIC_SYSFS: /* 0x62656572 local */
return "sysfs";
case S_MAGIC_SYSV2: /* 0x012FF7B6 local */
return "sysv2";
case S_MAGIC_SYSV4: /* 0x012FF7B5 local */
return "sysv4";
case S_MAGIC_TMPFS: /* 0x01021994 local */
return "tmpfs";
case S_MAGIC_TRACEFS: /* 0x74726163 local */
return "tracefs";
case S_MAGIC_UBIFS: /* 0x24051905 local */
return "ubifs";
case S_MAGIC_UDF: /* 0x15013346 local */
return "udf";
case S_MAGIC_UFS: /* 0x00011954 local */
return "ufs";
case S_MAGIC_UFS_BYTESWAPPED: /* 0x54190100 local */
return "ufs";
case S_MAGIC_USBDEVFS: /* 0x9FA2 local */
return "usbdevfs";
```

```

case S_MAGIC_V9FS: /* 0x01021997 local */
return "v9fs";
case S_MAGIC_VBOXSF: /* 0x786F4256 remote */
return "vboxsf";
case S_MAGIC_VMHGFS: /* 0xBACBACBC remote */
return "vmhgfs";
case S_MAGIC_VXFS: /* 0xA501FCF5 remote */
/* Veritas File System can run in single instance or clustered mode,
   so mark as remote to cater for the latter case. */
return "vxfs";
case S_MAGIC_VZFS: /* 0x565A4653 local */
return "vzfs";
case S_MAGIC_WSLFS: /* 0x53464846 local */
return "wslfs";
case S_MAGIC_XENFS: /* 0xABBA1974 local */
return "xenfs";
case S_MAGIC_XENIX: /* 0x012FF7B4 local */
return "xenix";
case S_MAGIC_XFS: /* 0x58465342 local */
return "xfs";
case S_MAGIC_XIAFS: /* 0x012FD16D local */
return "xia";
case S_MAGIC_Z3FOLD: /* 0x0033 local */
return "z3fold";
case S_MAGIC_ZFS: /* 0x2FC12FC1 local */
return "zfs";
case S_MAGIC_ZONEFS: /* 0x5A4F4653 local */
return "zonefs";
case S_MAGIC_ZSMALLOC: /* 0x58295829 local */
return "zsmallocfs";

# elif __GNU__
case FSTYPE_UFS:
return "ufs";
case FSTYPE_NFS:
return "nfs";
case FSTYPE_GFS:
return "gfs";
case FSTYPE_LFS:
return "lfs";
case FSTYPE_SYSV:
return "sysv";
case FSTYPE_FTP:
return "ftp";
case FSTYPE_TAR:
return "tar";
case FSTYPE_AR:
return "ar";
case FSTYPE_CPIO:
return "cpio";
case FSTYPE_MSLOSS:
return "msloss";
case FSTYPE_CPM:
return "cpm";
case FSTYPE_HFS:
return "hfs";
case FSTYPE_DTFS:
return "dtfs";
case FSTYPE_GRFS:

```

```

        return "grfs";
    case FSTYPE_TERM:
        return "term";
    case FSTYPE_DEV:
        return "dev";
    case FSTYPE_PROC:
        return "proc";
    case FSTYPE_IFSOCK:
        return "ifsock";
    case FSTYPE_AFS:
        return "afs";
    case FSTYPE_DFS:
        return "dfs";
    case FSTYPE_PROC9:
        return "proc9";
    case FSTYPE_SOCKET:
        return "socket";
    case FSTYPE_MISC:
        return "misc";
    case FSTYPE_EXT2FS:
        return "ext2/ext3";
    case FSTYPE_HTTP:
        return "http";
    case FSTYPE_MEMFS:
        return "memfs";
    case FSTYPE_ISO9660:
        return "iso9660";
#endif
default:
{
    unsigned long int type = statfsbuf->f_type;
    static char buf[sizeof "UNKNOWN (0x%lx)" - 3
                    + (sizeof type * CHAR_BIT + 3) / 4];
    sprintf (buf, "UNKNOWN (0x%lx)", type);
    return buf;
}
#endif
}

NODISCARD
static char *
human_access (struct stat const *statbuf)
{
    static char modebuf[12];
    filemodestring (statbuf, modebuf);
    modebuf[10] = 0;
    return modebuf;
}

NODISCARD
static char *
human_time (struct timespec t)
{
/* STR must be at least INT_BUFSIZE_BOUND (intmax_t) big, either
   because localtime_rz fails, or because the time zone is truly
   outlandish so that %z expands to a long string. */
    static char str[INT_BUFSIZE_BOUND (intmax_t)
                  + INT_STRLEN_BOUND (int) /* YYYY */
                  + 1 /* because YYYY might equal INT_MAX + 1900 */]

```

```

        + sizeof "-MM-DD HH:MM:SS.NNNNNNNNN +"];
static timezone_t tz;
if (!tz)
    tz = tzalloc (getenv ("TZ"));
struct tm tm;
int ns = t.tv_nsec;
if (localtime_rz (tz, &t.tv_sec, &tm))
    nstrftime (str, sizeof str, "%Y-%m-%d %H:%M:%S.%N %z", &tm, tz, ns);
else
{
    char sedbuf[INT_BUFSIZE_BOUND (intmax_t)];
    sprintf (str, "%s.%09d", timetostr (t.tv_sec, sedbuf), ns);
}
return str;
}

/* PFORMAT points to a '%' followed by a prefix of a format, all of
size PREFIX_LEN. The flags allowed for this format are
ALLOWED_FLAGS; remove other printf flags from the prefix, then
append SUFFIX. */
static void
make_format (char *pformat, size_t prefix_len, char const *allowed_flags,
            char const *suffix)
{
char *dst = pformat + 1;
char const *src;
char const *srclim = pformat + prefix_len;
for (src = dst; src < srclim && strchr (printf_flags, *src); src++)
    if (strchr (allowed_flags, *src))
        *dst++ = *src;
while (src < srclim)
    *dst++ = *src++;
strcpy (dst, suffix);
}

static void
out_string (char *pformat, size_t prefix_len, char const *arg)
{
make_format (pformat, prefix_len, "-", "s");
printf (pformat, arg);
}
static int
out_int (char *pformat, size_t prefix_len, intmax_t arg)
{
make_format (pformat, prefix_len, "-+ 0", "jd");
return printf (pformat, arg);
}
static int
out_uint (char *pformat, size_t prefix_len, uintmax_t arg)
{
make_format (pformat, prefix_len, "-0", "ju");
return printf (pformat, arg);
}
static void
out_uint_o (char *pformat, size_t prefix_len, uintmax_t arg)
{
make_format (pformat, prefix_len, "-#0", "jo");
printf (pformat, arg);
}
static void

```



```

        if (1 < w_d)
        {
            int w = w_d - precision;
            if (1 < w)
            {
                char *dst = pformat;
                for (char const *src = dst; src < p; src++)
                {
                    if (*src == '-')
                        frac_left_adjust = true;
                    else
                        *dst++ = *src;
                }
                sec_prefix_len =
                    (dst - pformat
                     + (frac_left_adjust ? 0 : sprintf (dst, "%d", w)));
            }
        }
    }

int divisor = 1;
for (int i = precision; i < 9; i++)
    divisor *= 10;
int frac_sec = arg.tv_nsec / divisor;
int int_len;

if (TYPE_SIGNED (time_t))
{
    bool minus_zero = false;
    if (arg.tv_sec < 0 && arg.tv_nsec != 0)
    {
        int frac_sec_modulus = 1000000000 / divisor;
        frac_sec = (frac_sec_modulus - frac_sec
                    - (arg.tv_nsec % divisor != 0));
        arg.tv_sec += (frac_sec != 0);
        minus_zero = (arg.tv_sec == 0);
    }
    int_len = (minus_zero
               ? out_minus_zero (pformat, sec_prefix_len)
               : out_int (pformat, sec_prefix_len, arg.tv_sec));
}
else
    int_len = out_uint (pformat, sec_prefix_len, arg.tv_sec);

if (precision)
{
    int prec = (precision < 9 ? precision : 9);
    int trailing_prec = precision - prec;
    int ilen = (int_len < 0 ? 0 : int_len);
    int trailing_width = (ilen < width && decimal_point_len < width - ilen
                           ? width - ilen - decimal_point_len - prec
                           : 0);
    printf ("%s%.*d%-*.*d", decimal_point, prec, frac_sec,
           trailing_width, trailing_prec, 0);
}
}

/* Print the context information of FILENAME, and return true iff the

```

```

context could not be obtained. */
NODISCARD
static bool
out_file_context (char *pformat, size_t prefix_len, char const *filename)
{
char *scontext;
bool fail = false;

if ((follow_links
    ? getfilecon (filename, &scontext)
    : lgetfilecon (filename, &scontext)) < 0)
{
    error (0, errno, _("failed to get security context of %s"),
        quoteaf (filename));
    scontext = nullptr;
    fail = true;
}
strcpy (pformat + prefix_len, "s");
printf (pformat, (scontext ? scontext : "?"));
if (scontext)
    freecon (scontext);
return fail;
}

/* Print statfs info. Return zero upon success, nonzero upon failure. */
NODISCARD
static bool
print_statfs (char *pformat, size_t prefix_len, MAYBE_UNUSED char mod, char m,
    int fd, char const *filename,
    void const *data)
{
STRUCT_STATVFS const *statfsbuf = data;
bool fail = false;

switch (m)
{
case 'n':
    out_string (pformat, prefix_len, filename);
    break;

case 'i':
{
#if STRUCT_STATXFS_F_FSID_IS_INTEGER
    uintmax_t fsid = statfsbuf->f_fsid;
    out_uint_x (pformat, prefix_len, fsid);
#endif
#endif
# elif __MVS__
;
#endif
# else
typedef unsigned int fsid_word;
static_assert (alignof (STRUCT_STATVFS) % alignof (fsid_word) == 0);
static_assert (offsetof (STRUCT_STATVFS, f_fsid) % alignof (fsid_word)
    == 0);
static_assert (sizeof statfsbuf->f_fsid % alignof (fsid_word) == 0);
fsid_word const *p = (fsid_word *) &statfsbuf->f_fsid;

/* Assume a little-endian word order, as that is compatible
with glibc's statvfs implementation. */
uintmax_t fsid = 0;
int words = sizeof statfsbuf->f_fsid / sizeof *p;
for (int i = 0; i < words && i * sizeof *p < sizeof fsid; i++)
}
}

```

```

    {
        uintmax_t u = p[words - 1 - i];
        fsid |= u << (i * CHAR_BIT * sizeof *p);
    }
    out_uint_x (pformat, prefix_len, fsid);
#endif
}

break;

case 'I':
OUT_NAMEMAX (pformat, prefix_len, SB_F_NAMEMAX (statfsbuf));
break;
case 't':
#if HAVE_STRUCT_STATXFS_F_TYPE
    out_uint_x (pformat, prefix_len, statfsbuf->f_type);
#else
    fputc ('?', stdout);
#endif
break;
case 'T':
out_string (pformat, prefix_len, human_fstype (statfsbuf));
break;
case 'b':
#ifndef __MVS__
    out_int (pformat, prefix_len, statfsbuf->f_blocks);
#endif
break;
case 'f':
#ifndef __MVS__
    out_int (pformat, prefix_len, statfsbuf->f_bfree);
#endif
break;
case 'a':
#ifndef __MVS__
    out_int (pformat, prefix_len, statfsbuf->f_bavail);
#endif
break;
case 's':
#ifndef __MVS__
    out_uint (pformat, prefix_len, statfsbuf->f_bsize);
#endif
break;
case 'S':
{
#ifndef __MVS__
    uintmax_t frsize = STATFS_FRSIZE (statfsbuf);
    if (!frsize)
        frsize = statfsbuf->f_bsiz;
    out_uint (pformat, prefix_len, frsize);
#endif
}
break;
case 'c':
#ifndef __MVS__
    out_uint (pformat, prefix_len, statfsbuf->f_files);
#endif
break;
case 'd':
#ifndef __MVS__
    out_int (pformat, prefix_len, statfsbuf->f_ffree);
#endif
}

```

```

#endif
    break;
default:
    fputc ('?', stdout);
    break;
}
return fail;
}

/* Return any bind mounted source for a path.
The caller should not free the returned buffer.
Return nullptr if no bind mount found. */
NODISCARD
static char const *
find_bind_mount (char const * name)
{
    char const * bind_mount = nullptr;

    static struct mount_entry *mount_list;
    static bool tried_mount_list = false;
    if (!tried_mount_list) /* attempt/warn once per process. */
    {
        if (!(mount_list = read_file_system_list (false)))
            error (0, errno, "%s", _("cannot read table of mounted file systems"));
        tried_mount_list = true;
    }

    struct stat name_stats;
    if (stat (name, &name_stats) != 0)
        return nullptr;

    struct mount_entry *me;
    for (me = mount_list; me; me = me->me_next)
    {
        if (me->me_dummy && me->me_devname[0] == '/'
            && STREQ (me->me_mountdir, name))
        {
            struct stat dev_stats;

            if (stat (me->me_devname, &dev_stats) == 0
                && psame_inode (&name_stats, &dev_stats))
            {
                bind_mount = me->me_devname;
                break;
            }
        }
    }

    return bind_mount;
}

/* Print mount point. Return zero upon success, nonzero upon failure. */
NODISCARD
static bool
out_mount_point (char const *filename, char *pformat, size_t prefix_len,
                 const struct stat *statp)
{
    char const *np = "?", *bp = nullptr;
    char *mp = nullptr;

```

```

bool fail = true;

/* Look for bind mounts first. Note we output the immediate alias,
   rather than further resolving to a base device mount point. */
if (follow_links || !S_ISLNK (statp->st_mode))
{
    char *resolved = canonicalize_file_name (filename);
    if (!resolved)
    {
        error (0, errno, _("failed to canonicalize %s"), quoteaf (filename));
        goto print_mount_point;
    }
    bp = find_bind_mount (resolved);
    free (resolved);
    if (bp)
    {
        fail = false;
        goto print_mount_point;
    }
}

/* If there is no direct bind mount, then navigate
   back up the tree looking for a device change.
   Note we don't detect if any of the directory components
   are bind mounted to the same device, but that's OK
   since we've not directly queried them. */
if ((mp = find_mount_point (filename, statp)))
{
    /* This dir might be bind mounted to another device,
       so we resolve the bound source in that case also. */
    bp = find_bind_mount (mp);
    fail = false;
}

print_mount_point:

out_string (pformat, prefix_len, bp ? bp : mp ? mp : np);
free (mp);
return fail;
}

/* Map a TS with negative TS.tv_nsec to {0,0}. */
static inline struct timespec
neg_to_zero (struct timespec ts)
{
if (0 <= ts.tv_nsec)
    return ts;
struct timespec z = {0};
return z;
}

/* Set the quoting style default if the environment variable
QUOTING_STYLE is set. */

static void
getenv_quoting_style (void)
{
char const *q_style = getenv ("QUOTING_STYLE");
if (q_style)
{

```

```

int i = ARGMATCH (q_style, quoting_style_args, quoting_style_vals);
if (0 <= i)
    set_quoting_style (nullptr, quoting_style_vals[i]);
else
{
    set_quoting_style (nullptr, shell_escape_always_quoting_style);
    error (0, 0, _("ignoring invalid value of environment "
                  "variable QUOTING_STYLE: %s"), quote (q_style));
}
}
else
    set_quoting_style (nullptr, shell_escape_always_quoting_style);
}

/* Equivalent to quotearg(), but explicit to avoid syntax checks. */
#define quoteN(x) quotearg_style (get_quoting_style (nullptr), x)

/* Output a single-character \ escape. */

static void
print_esc_char (char c)
{
switch (c)
{
case 'a':                      /* Alert. */
c = '\a';
break;
case 'b':                      /* Backspace. */
c = '\b';
break;
case 'e':                      /* Escape. */
c = '\x1B';
break;
case 'f':                      /* Form feed. */
c = '\f';
break;
case 'n':                      /* New line. */
c = '\n';
break;
case 'r':                      /* Carriage return. */
c = '\r';
break;
case 't':                      /* Horizontal tab. */
c = '\t';
break;
case 'v':                      /* Vertical tab. */
c = '\v';
break;
case '\"':
case '\\':
break;
default:
error (0, 0, _("warning: unrecognized escape '\\%c'"), c);
break;
}
putchar (c);
}

ATTRIBUTE_PURE
static size_t

```

```

format_code_offset (char const *directive)
{
size_t len = strspn (directive + 1, printf_flags);
char const *fmt_char = directive + len + 1;
fmt_char += strspn (fmt_char, digits);
if (*fmt_char == '.')
    fmt_char += 1 + strspn (fmt_char + 1, digits);
return fmt_char - directive;
}

/* Print the information specified by the format string, FORMAT,
calling PRINT_FUNC for each %-directive encountered.
Return zero upon success, nonzero upon failure. */
NODISCARD
static bool
print_it (char const *format, int fd, char const *filename,
    bool (*print_func) (char *, size_t, char, char,
                        int, char const *, void const *),
    void const *data)
{
bool fail = false;

/* Add 2 to accommodate our conversion of the stat '%s' format string
   to the longer printf '%llu' one. */
enum
{
MAX_ADDITIONAL_BYTES =
(MAX (sizeof "jd",
      MAX (sizeof "jo", MAX (sizeof "ju", sizeof "jx"))))
 - 1)
};

size_t n_alloc = strlen (format) + MAX_ADDITIONAL_BYTES + 1;
char *dest = xmalloc (n_alloc);
char const *b;
for (b = format; *b; b++)
{
switch (*b)
{
case '%':
{
size_t len = format_code_offset (b);
char fmt_char = *(b + len);
char mod_char = 0;
memcpy (dest, b, len);
b += len;

switch (fmt_char)
{
case '\0':
--b;
FALLTHROUGH;
case '%':
if (1 < len)
{
dest[len] = fmt_char;
dest[len + 1] = '\0';
error (EXIT_FAILURE, 0, _("'%s: invalid directive"),
quote (dest));
}
putchar ('%');
}
}
}
}

```

```

        break;
case 'H':
case 'L':
    mod_char = fmt_char;
    fmt_char = *(b + 1);
    if (print_func == print_stat
        && (fmt_char == 'd' || fmt_char == 'r'))
    {
        b++;
    }
    else
    {
        fmt_char = mod_char;
        mod_char = 0;
    }
FALLTHROUGH;
default:
    fail |= print_func (dest, len, mod_char, fmt_char,
                        fd, filename, data);
    break;
}
break;
}

case '\\':
if (!interpret_backslash_escapes)
{
    putchar ('\\');
    break;
}
++b;
if (isodigit (*b))
{
    int esc_value = octtobin (*b);
    int esc_length = 1; /* number of octal digits */
    for (++b; esc_length < 3 && isodigit (*b);
         ++esc_length, ++b)
    {
        esc_value = esc_value * 8 + octtobin (*b);
    }
    putchar (esc_value);
    --b;
}
else if (*b == 'x' && c_isxdigit (to_uchar (b[1])))
{
    int esc_value = hextobin (b[1]); /* Value of \xhh escape. */
    /* A hexadecimal \xhh escape sequence must have
     * 1 or 2 hex. digits. */
    ++b;
    if (c_isxdigit (to_uchar (b[1])))
    {
        ++b;
        esc_value = esc_value * 16 + hextobin (*b);
    }
    putchar (esc_value);
}
else if (*b == '\\0')
{
    error (0, 0, _("warning: backslash at end of format"));
    putchar ('\\');
}
```

```

        /* Arrange to exit the loop. */
        --b;
    }
} else
{
    print_esc_char (*b);
}
break;

default:
putchar (*b);
break;
}
}
free (dest);

fputs (trailing_delim, stdout);

return fail;
}

/* Stat the file system and print what we find. */
NODISCARD
static bool
do_statfs (char const *filename, char const *format)
{
STRUCT_STATVFS statfsbuf;

if (STREQ (filename, "-"))
{
    error (0, 0, _("using %s to denote standard input does not work"
                  " in file system mode"), quoteaf (filename));
    return false;
}

#ifndef __MVS__
if (STATFS (filename, &statfsbuf) != 0)
{
    error (0, errno, _("cannot read file system information for %s"),
          quoteaf (filename));
    return false;
}
#endif

bool fail = print_it (format, -1, filename, print_statfs, &statfsbuf);
return ! fail;
}

struct print_args {
struct stat *st;
struct timespec btime;
};

/* Ask statx to avoid syncing? */
static bool dont_sync;

/* Ask statx to force sync? */
static bool force_sync;

#if USE_STATX

```

```

static unsigned int
fmt_to_mask (char fmt)
{
switch (fmt)
{
case 'N':
return STATX_MODE;
case 'd':
case 'D':
return STATX_MODE;
case 'i':
return STATX_INO;
case 'a':
case 'A':
return STATX_MODE;
case 'f':
return STATX_MODE|STATX_TYPE;
case 'F':
return STATX_TYPE;
case 'h':
return STATX_NLINK;
case 'u':
case 'U':
return STATX_UID;
case 'g':
case 'G':
return STATX_GID;
case 'm':
return STATX_MODE|STATX_INO;
case 's':
return STATX_SIZE;
case 't':
case 'T':
return STATX_MODE;
case 'b':
return STATX_BLOCKS;
case 'w':
case 'W':
return STATX_BTIME;
case 'x':
case 'X':
return STATX_ATIME;
case 'y':
case 'Y':
return STATX_MTIME;
case 'z':
case 'Z':
return STATX_CTIME;
}
return 0;
}

```

```

ATTRIBUTE_PURE
static unsigned int
format_to_mask (char const *format)
{
unsigned int mask = 0;
char const *b;

for (b = format; *b; b++)

```

```

{
if (*b != '%')
    continue;

b += format_code_offset (b);
if (*b == '\0')
    break;
mask |= fmt_to_mask (*b);
}
return mask;
}

/* statx the file and print what we find */
NODISCARD
static bool
do_stat (char const *filename, char const *format, char const *format2)
{
int fd = STREQ (filename, "-") ? 0 : AT_FDCWD;
int flags = 0;
struct stat st;
struct statx stx = {0};
char const *pathname = filename;
struct print_args pa;
pa.st = &st;
pa.btime = (struct timespec) { .tv_sec = -1, .tv_nsec = -1 };

if (AT_FDCWD != fd)
{
    pathname = "";
    flags = AT_EMPTY_PATH;
}
else if (!follow_links)
{
    flags = AT_SYMLINK_NOFOLLOW;
}

if (dont_sync)
    flags |= AT_STATX_DONT_SYNC;
else if (force_sync)
    flags |= AT_STATX_FORCE_SYNC;

if (! force_sync)
    flags |= AT_NO_AUTOMOUNT;

fd = statx (fd, pathname, flags, format_to_mask (format), &stx);
if (fd < 0)
{
    if (flags & AT_EMPTY_PATH)
        error (0, errno, _("cannot stat standard input"));
    else
        error (0, errno, _("cannot statx %s"), quoteaf (filename));
    return false;
}

if (S_ISBLK (stx.stx_mode) || S_ISCHR (stx.stx_mode))
    format = format2;

statx_to_stat (&stx, &st);
if (stx.stx_mask & STATX_BTIME)
    pa.btime = statx_timestamp_to_timespec (stx.stx_btime);

```

```

bool fail = print_it (format, fd, filename, print_stat, &pa);
return ! fail;
}

#else /* USE_STATX */

static struct timespec
get_birthtime (int fd, char const *filename, struct stat const *st)
{
struct timespec ts = get_stat_birthtime (st);

#if HAVE_GETATTRATR
if (ts.tv_nsec < 0)
{
nvlist_t *response;
if ((fd < 0
    ? setattrat (AT_FDCWD, XATTR_VIEW_READWRITE, filename, &response)
    : fgetattr (fd, XATTR_VIEW_READWRITE, &response))
== 0)
{
uint64_t *val;
uint_t n;
if (nvlist_lookup_uint64_array (response, A_CRTIME, &val, &n) == 0
    && 2 <= n
    && val[0] <= TYPE_MAXIMUM (time_t)
    && val[1] < 1000000000 * 2 /* for leap seconds */
    {
ts.tv_sec = val[0];
ts.tv_nsec = val[1];
}
nvlist_free (response);
}
}
#endif
}

/* stat the file and print what we find */
NODISCARD
static bool
do_stat (char const *filename, char const *format,
         char const *format2)
{
int fd = STREQ (filename, "-") ? 0 : -1;
struct stat statbuf;
struct print_args pa;
pa.st = &statbuf;
pa.btime = (struct timespec) { .tv_sec = -1, .tv_nsec = -1 };

if (0 <= fd)
{
if (fstat (fd, &statbuf) != 0)
{
error (0, errno, _("cannot stat standard input"));
return false;
}
}

```

```

/* We can't use the shorter
   (follow_links?stat:Istat) (filename, &statbug)
   since stat might be a function-like macro. */
else if ((follow_links
    ? stat (filename, &statbuf)
    : Istat (filename, &statbuf)) != 0)
{
    error (0, errno, _("cannot stat %s"), quoteaf (filename));
    return false;
}

if (S_ISBLK (statbuf.st_mode) || S_ISCHR (statbuf.st_mode))
    format = format2;

bool fail = print_it (format, fd, filename, print_stat, &pa);
return ! fail;
}
#endif /* USE_STATX */

/* POSIX requires 'ls' to print file sizes without a sign, even
   when negative. Be consistent with that. */

static uintmax_t
unsigned_file_size (off_t size)
{
    return size + (size < 0) * ((uintmax_t) OFF_T_MAX - OFF_T_MIN + 1);
}

/* Print stat info. Return zero upon success, nonzero upon failure. */
static bool
print_stat (char *pformat, size_t prefix_len, char mod, char m,
            int fd, char const *filename, void const *data)
{
    struct print_args *parg = (struct print_args *) data;
    struct stat *statbuf = parg->st;
    struct timespec btime = parg->btime;
    struct passwd *pw_ent;
    struct group *gw_ent;
    bool fail = false;

    switch (m)
    {
        case 'n':
            out_string (pformat, prefix_len, filename);
            break;
        case 'N':
            out_string (pformat, prefix_len, quoteN (filename));
            if (S_ISLNK (statbuf->st_mode))
            {
                char *linkname = areadlink_with_size (filename, statbuf->st_size);
                if (linkname == nullptr)
                {
                    error (0, errno, _("cannot read symbolic link %s"),
                           quoteaf (filename));
                    return true;
                }
                printf (" -> ");
                out_string (pformat, prefix_len, quoteN (linkname));
                free (linkname);
            }
    }
}

```

```

break;
case 'd':
if (mod == 'H')
    out_uint (pformat, prefix_len, major (statbuf->st_dev));
else if (mod == 'L')
    out_uint (pformat, prefix_len, minor (statbuf->st_dev));
else
    out_uint (pformat, prefix_len, statbuf->st_dev);
break;
case 'D':
out_uint_x (pformat, prefix_len, statbuf->st_dev);
break;
case 'i':
out_uint (pformat, prefix_len, statbuf->st_ino);
break;
case 'a':
out_uint_o (pformat, prefix_len, statbuf->st_mode & CHMOD_MODE_BITS);
break;
case 'A':
out_string (pformat, prefix_len, human_access (statbuf));
break;
case 'f':
out_uint_x (pformat, prefix_len, statbuf->st_mode);
break;
case 'F':
out_string (pformat, prefix_len, file_type (statbuf));
break;
case 'h':
out_uint (pformat, prefix_len, statbuf->st_nlink);
break;
case 'u':
out_uint (pformat, prefix_len, statbuf->st_uid);
break;
case 'U':
pw_ent = getpwuid (statbuf->st_uid);
out_string (pformat, prefix_len,
            pw_ent ? pw_ent->pw_name : "UNKNOWN");
break;
case 'g':
out_uint (pformat, prefix_len, statbuf->st_gid);
break;
case 'G':
gw_ent = getgrgid (statbuf->st_gid);
out_string (pformat, prefix_len,
            gw_ent ? gw_ent->gr_name : "UNKNOWN");
break;
case 'm':
fail |= out_mount_point (filename, pformat, prefix_len, statbuf);
break;
case 's':
out_uint (pformat, prefix_len, unsigned_file_size (statbuf->st_size));
break;
case 'r':
if (mod == 'H')
    out_uint (pformat, prefix_len, major (statbuf->st_rdev));
else if (mod == 'L')
    out_uint (pformat, prefix_len, minor (statbuf->st_rdev));
else
    out_uint (pformat, prefix_len, statbuf->st_rdev);
break;

```

```

case 'R':
out_uint_x (pformat, prefix_len, statbuf->st_rdev);
break;
case 't':
out_uint_x (pformat, prefix_len, major (statbuf->st_rdev));
break;
case 'T':
out_uint_x (pformat, prefix_len, minor (statbuf->st_rdev));
break;
case 'B':
out_uint (pformat, prefix_len, ST_NBLOCKSIZE);
break;
case 'b':
out_uint (pformat, prefix_len, STP_NBLOCKS (statbuf));
break;
case 'o':
out_uint (pformat, prefix_len, STP_BLKSIZE (statbuf));
break;
case 'w':
{
#endif ! USE_STATX
    btime = get_birthtime (fd, filename, statbuf);
#endif
    if (btime.tv_nsec < 0)
        out_string (pformat, prefix_len, "-");
    else
        out_string (pformat, prefix_len, human_time (btime));
}
break;
case 'W':
{
#endif ! USE_STATX
    btime = get_birthtime (fd, filename, statbuf);
#endif
    out_epoch_sec (pformat, prefix_len, neg_to_zero (btime));
}
break;
case 'x':
out_string (pformat, prefix_len, human_time (get_stat_atime (statbuf)));
break;
case 'X':
out_epoch_sec (pformat, prefix_len, get_stat_atime (statbuf));
break;
case 'y':
out_string (pformat, prefix_len, human_time (get_stat_mtime (statbuf)));
break;
case 'Y':
out_epoch_sec (pformat, prefix_len, get_stat_mtime (statbuf));
break;
case 'z':
out_string (pformat, prefix_len, human_time (get_stat_ctime (statbuf)));
break;
case 'Z':
out_epoch_sec (pformat, prefix_len, get_stat_ctime (statbuf));
break;
case 'C':
fail |= out_file_context (pformat, prefix_len, filename);
break;
default:
    fputc ('?', stdout);
}

```

```

        break;
    }
return fail;
}

/* Return an allocated format string in static storage that
corresponds to whether FS and TERSE options were declared. */
static char *
default_format (bool fs, bool terse, bool device)
{
char *format;
if (fs)
{
    if (terse)
        format = xstrdup (fmt_terse_fs);
    else
    {
        /* TRANSLATORS: This string uses format specifiers from
         'stat --help' with --file-system, and NOT from printf. */
        format = xstrdup (_(" File: \"%n\"\n"
                            "   ID: %-8i Namelen: %-7i Type: %T\n"
                            "Block size: %-10s Fundamental block size: %S\n"
                            "Blocks: Total: %-10b Free: %-10f Available: %a\n"
                            "Inodes: Total: %-10c Free: %d\n"));
    }
}
else /* ! fs */
{
    if (terse)
    {
        if (0 < is_selinux_enabled ())
            format = xstrdup (fmt_terse_selinux);
        else
            format = xstrdup (fmt_terse_regular);
    }
else
{
    char *temp;
    /* TRANSLATORS: This string uses format specifiers from
     'stat --help' without --file-system, and NOT from printf. */
    format = xstrdup (_("\
File: %N\n\
Size: %-10s\tBlocks: %-10b IO Block: %-6o %F\n\
"));

    temp = format;
    if (device)
    {
        /* TRANSLATORS: This string uses format specifiers from
         'stat --help' without --file-system, and NOT from printf. */
        format = xasprintf ("%s%s", format, _("\
" "Device: %Hd,%Ld\lInode: %-10i Links: %-5h Device type: %Hr,%Lr\n\
")));
    }
else
{
        /* TRANSLATORS: This string uses format specifiers from
         'stat --help' without --file-system, and NOT from printf. */
        format = xasprintf ("%s%s", format, _("\
" "Device: %Hd,%Ld\lInode: %-10i Links: %h\n\
"));
}
}
}

```

```

    ");
    }
    free (temp);

    temp = format;
    /* TRANSLATORS: This string uses format specifiers from
       'stat --help' without --file-system, and NOT from printf. */
    format = xasprintf ("%s%s", format, _("\
" "Access: (%04a/%10.10A) Uid: (%5u/%8U) Gid: (%5g/%8G)\n\
"));
    free (temp);

    if (0 < is_selinux_enabled ())
    {
        temp = format;
        /* TRANSLATORS: This string uses format specifiers from
           'stat --help' without --file-system, and NOT from printf. */
        format = xasprintf ("%s%s", format, _("Context: %C\n"));
        free (temp);
    }

    temp = format;
    /* TRANSLATORS: This string uses format specifiers from
       'stat --help' without --file-system, and NOT from printf. */
    format = xasprintf ("%s%s", format,
        _("Access: %x\n"
          "Modify: %y\n"
          "Change: %z\n"
          " Birth: %w\n"));
    free (temp);
}
}

return format;
}

void
usage (int status)
{
if (status != EXIT_SUCCESS)
    emit_try_help ();
else
{
    printf (_("Usage: %s [OPTION]... FILE...\n"), program_name);
    fputs (_("\
Display file or file system status.\n\
"), stdout);

    emit_mandatory_arg_note ();

    fputs (_("\
-L, --dereference   follow links\n\
-f, --file-system   display file system status instead of file status\n\
"), stdout);
    fputs (_("\
--cached=MODE   specify how to use cached attributes;\n\
                  useful on remote file systems. See MODE below\n\
"), stdout);
    fputs (_("\
-c --format=FORMAT  use the specified FORMAT instead of the default;\n\
                  output a newline after each use of FORMAT\n\
"), stdout);
}
}

```

```

--printf=FORMAT like --format, but interpret backslash escapes,\n\
    and do not output a mandatory trailing newline;\n\
    if you want a newline, include \\n in FORMAT\n\
-t, --terse      print the information in terse form\n\
"), stdout);
    fputs (HELP_OPTION_DESCRIPTION, stdout);
    fputs (VERSION_OPTION_DESCRIPTION, stdout);

    fputs (_("\n\
The MODE argument of --cached can be: always, never, or default.\n\
'always' will use cached attributes if available, while\n\
'never' will try to synchronize with the latest attributes, and\n\
'default' will leave it up to the underlying file system.\n\
"), stdout);

    fputs (_("\n\
The valid format sequences for files (without --file-system):\n\
\n\
%a permission bits in octal (see '#' and '0' printf flags)\n\
%A permission bits and file type in human readable form\n\
%b number of blocks allocated (see %B)\n\
%B the size in bytes of each block reported by %b\n\
%C SELinux security context string\n\
"), stdout);
    fputs (_("\
%d device number in decimal (st_dev)\n\
%D device number in hex (st_dev)\n\
%Hd major device number in decimal\n\
%Ld minor device number in decimal\n\
%f raw mode in hex\n\
%F file type\n\
%g group ID of owner\n\
%G group name of owner\n\
"), stdout);
    fputs (_("\
%h number of hard links\n\
%i inode number\n\
%m mount point\n\
%n file name\n\
%N quoted file name with dereference if symbolic link\n\
%o optimal I/O transfer size hint\n\
%s total size, in bytes\n\
%r device type in decimal (st_rdev)\n\
%R device type in hex (st_rdev)\n\
%Hr major device type in decimal, for character/block device special files\n\
%Lr minor device type in decimal, for character/block device special files\n\
%tr major device type in hex, for character/block device special files\n\
%Tr minor device type in hex, for character/block device special files\n\
"), stdout);
    fputs (_("\
%u user ID of owner\n\
%U user name of owner\n\
%w time of file birth, human-readable; - if unknown\n\
%W time of file birth, seconds since Epoch; 0 if unknown\n\
%x time of last access, human-readable\n\
%X time of last access, seconds since Epoch\n\
%y time of last data modification, human-readable\n\
%Y time of last data modification, seconds since Epoch\n\
%z time of last status change, human-readable\n\
%Z time of last status change, seconds since Epoch\n\

```

```

\n\
"), stdout);

    fputs (_("\
Valid format sequences for file systems:\n\
\n\
%a free blocks available to non-superuser\n\
%b total data blocks in file system\n\
%c total file nodes in file system\n\
%d free file nodes in file system\n\
%f free blocks in file system\n\
"), stdout);
    fputs (_("\
%i file system ID in hex\n\
%l maximum length of filenames\n\
%n file name\n\
%s block size (for faster transfers)\n\
%S fundamental block size (for block counts)\n\
%t file system type in hex\n\
%T file system type in human readable form\n\
"), stdout);

    printf (_("\n\
--terse is equivalent to the following FORMAT:\n\
%s\
"),
#if HAVE_SELINUX_SELINUX_H
    fmt_terse_selinux
#else
    fmt_terse_regular
#endif
);

    printf (_("\
--terse --file-system is equivalent to the following FORMAT:\n\
%s\
"), fmt_terse_fs);

    printf (USAGE_BUILTIN_WARNING, PROGRAM_NAME);
    emit_ancillary_info (PROGRAM_NAME);
}
exit (status);
}

int
main (int argc, char *argv[])
{
int c;
bool fs = false;
bool terse = false;
char *format = nullptr;
char *format2;
bool ok = true;

initialize_main (&argc, &argv);
set_program_name (argv[0]);
setlocale (LC_ALL, "");
bindtextdomain (PACKAGE, LOCALEDIR);
textdomain (PACKAGE);

```

```

struct Iconv const *locale = localeconv ();
decimal_point = (locale->decimal_point[0] ? locale->decimal_point : ".");
decimal_point_len = strlen (decimal_point);

atexit (close_stdout);

while ((c = getopt_long (argc, argv, "c:fLt", long_options, nullptr)) != -1)
{
    switch (c)
    {
        case PRINTF_OPTION:
            format = optarg;
            interpret_backslash_escapes = true;
            trailing_delim = "";
            break;

        case 'c':
            format = optarg;
            interpret_backslash_escapes = false;
            trailing_delim = "\n";
            break;

        case 'L':
            follow_links = true;
            break;

        case 'f':
            fs = true;
            break;

        case 't':
            terse = true;
            break;

        case 0:
            switch (XARGMATCH ("--cached", optarg, cached_args, cached_modes))
            {
                case cached_never:
                    force_sync = true;
                    dont_sync = false;
                    break;
                case cached_always:
                    force_sync = false;
                    dont_sync = true;
                    break;
                case cached_default:
                    force_sync = false;
                    dont_sync = false;
            }
            break;

        case_GETOPT_HELP_CHAR;

        case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

        default:
            usage (EXIT_FAILURE);
    }
}

```

```

if (argc == optind)
{
    error (0, 0, _("missing operand"));
    usage (EXIT_FAILURE);
}

if (format)
{
    if (strstr (format, "%N"))
        getenv_quoting_style ();
    format2 = format;
}
else
{
    format = default_format (fs, terse, /* device= */ false);
    format2 = default_format (fs, terse, /* device= */ true);
}

for (int i = optind; i < argc; i++)
    ok &= (fs
        ? do_statfs (argv[i], format)
        : do_stat (argv[i], format, format2));

main_exit (ok ? EXIT_SUCCESS : EXIT_FAILURE);
}

"""

,
"patch":
"""

diff --git a/src/stat.c b/src/stat.c
index dd86450..4c9aaf5 100644
--- a/src/stat.c
+++ b/src/stat.c
@@ -98,7 +98,14 @@
# elif HAVE_STRUCT_STATFS_F_NAMEMAX
# define SB_F_NAMEMAX(S) ((S)->f_namemax)
# endif
-# define STATFS statfs
+
+#ifdef __MVS__
+#include <sys/statfs.h>
+# define STATFS w_statfs
+#else
+ # define STATFS statfs
+#endif
+
# if HAVE_OS_H /* BeOS */
/* BeOS has a statvfs function, but it does not return sensible values
   for f_files, f_ffree and f_favail, and lacks f_type, f_basetype and
@@ -132,7 +139,11 @@
# define STRUCT_STATXFS_F_FSID_IS_INTEGER true
# define STATFS_FRSIZE(S) ((S)->block_size)
# else
+#ifndef __MVS__
# define STRUCT_STATVFS struct statfs
+#else
+# define STRUCT_STATVFS struct w_statfs
+#endif
# define STRUCT_STATXFS_F_FSID_IS_INTEGER
STRUCT_STATFS_F_FSID_IS_INTEGER
# if HAVE_STRUCT_STATFS_F_FRSIZE

```

```

# define STATFS_FRSIZE(S) ((S)->f_frsize)
@@ -257,8 +268,12 @@ NODISCARD
static char const *
human_fstype (STRUCT_STATVFS const *statfsbuf)
{
-#ifdef STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME
+ifdef STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME
+#if defined __MVS__
+ return 0;
+#else
return statfsbuf->STATXFS_FILE_SYSTEM_TYPE_MEMBER_NAME;
+#endif
#else
switch (statfsbuf->f_type)
{
@@ -873,6 +888,9 @@ print_statfs (char *pformat, size_t prefix_len, MAYBE_UNUSED char
mod, char m,
{
#if STRUCT_STATXFS_F_FSID_IS_INTEGER
    uintmax_t fsid = statfsbuf->f_fsid;
+    out_uint_x (pformat, prefix_len, fsid);
+#elif __MVS__
+    ;
#else
    typedef unsigned int fsid_word;
    static_assert (alignof (STRUCT_STATVFS) % alignof (fsid_word) == 0);
@@ -890,8 +908,8 @@ print_statfs (char *pformat, size_t prefix_len, MAYBE_UNUSED char
mod, char m,
        uintmax_t u = p[words - 1 - i];
        fsid |= u << (i * CHAR_BIT * sizeof *p);
    }
#endif
    out_uint_x (pformat, prefix_len, fsid);
#endif
}
break;

@@ -909,30 +927,44 @@ print_statfs (char *pformat, size_t prefix_len, MAYBE_UNUSED
char mod, char m,
    out_string (pformat, prefix_len, human_fstype (statfsbuf));
    break;
    case 'b':
+ifndef __MVS__
        out_int (pformat, prefix_len, statfsbuf->f_blocks);
+endif
        break;
    case 'f':
+ifndef __MVS__
        out_int (pformat, prefix_len, statfsbuf->f_bfree);
+endif
        break;
    case 'a':
+ifndef __MVS__
        out_int (pformat, prefix_len, statfsbuf->f_bavail);
+endif
        break;
    case 's':
+ifndef __MVS__
        out_uint (pformat, prefix_len, statfsbuf->f_bsize);
+endif

```

```

        break;
    case 'S':
    {
#ifndef __MVS__
        uintmax_t frsize = STATFS_FRSIZE (statfsbuf);
        if (!frsize)
            frsize = statfsbuf->f_bsize;
        out_uint (pformat, prefix_len, frsize);
#endif
    }
    break;
    case 'c':
#ifndef __MVS__
    out_uint (pformat, prefix_len, statfsbuf->f_files);
#endif
    break;
    case 'd':
#ifndef __MVS__
    out_int (pformat, prefix_len, statfsbuf->f_ffree);
#endif
    break;
    default:
        fputc ('?', stdout);
@@ -1267,12 +1299,14 @@ do_statfs (char const *filename, char const *format)
    return false;
}

#ifndef __MVS__
if (STATFS (filename, &statfsbuf) != 0)
{
    error (0, errno, _("cannot read file system information for %s"),
           quoteaf (filename));
    return false;
}
#endif

bool fail = print_it (format, -1, filename, print_statfs, &statfsbuf);
return !fail;
"""
}

]

```