
```

::::::::::::: perror_demo.c :::::::::::::::
#include      <stdio.h>
#include      <errno.h>
#include      <string.h>
#include      <fcntl.h>

/* program    perror_demo.c
 * purpose    show how errno is set by syscalls
 *            and how perror prints the associated message
 * usage      perror_demo
 * action     tries to open() a file for a given mode
 *            then, if error, prints errno and message
 */

int main()
{
    char    name[100];          /* filename */
    int     mode;               /* second arg to open */
    int     rv = 0;             /* program status reprt */

    printf("Open what file or dir? ");
    fgets(name, 100, stdin);
    name[strlen(name)-1] = '\0';
    printf("    Open %s for 0,1, or 2? ", name);
    scanf("%d", &mode);
    if ( open( name, mode ) == -1 ){
        printf("Oh No!, errno %d just occurred\n", errno);
        perror( name );                /* print message */
        rv = 1;
    }
    else
        printf("%s opened ok\n", name );

    return rv;
}

::::::::::::: dig :::::::::::::::
#!/bin/sh
#
# script    dig
# purpose   show how to dig deep holes in a file system
# method    mkdir s; cd s; repeat (like on shampoo bottles)
# warning   good way to annoy system managers

    while true
    do
        mkdir s
        cd s
    done

::::::::::::: rml.c :::::::::::::::
#include      <stdio.h>
#include      <unistd.h>

/* example of using unlink to 'delete' a file */

int main(int ac, char *av[])
{
    int     rv = 0;
    while(--ac){
        if ( unlink( **++av ) == -1 ){
            perror(*av);
            rv = 1;
        }
        else
            printf("%s is gone\n", *av);
    }
    return rv;
}

```

```
::::::::::::: rm2.c :::::::::::::::
#include      <stdio.h>
#include      <unistd.h>
/*
 * rm2 - interactive version of rm using unlink(2)
 */

int main()
{
    char    name[100];
    char    confirm[100];
    int     rv = 0;                /* return value from main */

    while ( printf("delete what file? "), fgets(name,100,stdin) ){
        printf("about to unlink %s, ok? ", name );
        fgets(confirm,100,stdin);
        if ( *confirm == 'y' ){
            if( unlink(name) == -1 ){
                perror(name);
                rv = 1;
            }
            else
                printf("%s is gone\n", name);
        }
    }
    return rv;
}

::::::::::::: mv1.c :::::::::::::::
#include      <stdio.h>
#include      <errno.h>
#include      <stdlib.h>
#include      <unistd.h>

/*
 * program mv1.c
 * purpose show how to use link and unlink to rename a file
 * notes checks errno to make it more adaptable
 * note This version could be replaced by rename() but it
 * works differently when the target exists
 */

int main(int ac, char *av[])
{
    if ( link( av[1], av[2]) != -1 )
        unlink( av[1] );

    else if ( errno == EEXIST )        /* name2 already in use */
    {
        if (unlink( av[2] ) == -1 )    /* not any more          */
        {
            perror(av[2]);
            exit(1);
        }
        return main(ac,av);           /* and try again          */
    }
    return 0;
}


```

```
..... mv2.c .....
#include <stdio.h>
#include <limits.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

/**
**      mv28 version 1
**
**      if last arg is a dir, then move all files into there
**      if two args and last is a file, then move first to
**      second. will clobber second if exists.
**      Uses isadir .
**
**      problems: 1) what if an arg is a dir?
**                  2) what about paths in source files?
**/

void mv(char *,char *);
void mv_to_newdir(int, char **, char *);
int isadir(char *);

int main( int ac, char *av[] )
{
    if ( ac < 3 ){
        fprintf( stderr, "too few args\n");
        exit(1);
    }

    /*
    *      if last arg is a dir, then move args to there
    */
    if ( isadir( av[ac-1] ) )
        mv_to_newdir(ac-1,av,av[ac-1]);

    /*
    *      last arg is not a dir, so must be two args only
    */
    else if ( ac != 3 ){
        fprintf( stderr, "too many args\n");
        exit(1);
    }
    else mv(av[1], av[2]);
    return 0;
}

void mv_to_newdir( int ac, char *av[], char *newdir )
/*
*      move av[1], av[2], ... av[ac-1] into newdir
*/
{
    char    newpath[PATH_MAX];
    int     pos ;

    for( pos = 1 ; pos<ac; pos++ ){
        if ( strlen(newdir)+strlen(av[pos])+2 <= PATH_MAX )
        {
            sprintf( newpath , "%s/%s", newdir, av[pos]);
            mv( av[pos] , newpath );
        }
    }
}

void mv( char *oldname, char *newname )
{
    if ( link(oldname, newname) == -1 || unlink(oldname) == -1 ){
        fprintf(stderr,"error mv'ing %s to %s\n", oldname, newname);
        exit(1);
    }
}


```

```

::::::::::::: pwd28.c :::::::::::::::
#include      <stdio.h>
#include      <sys/types.h>
#include      <sys/stat.h>
#include      <dirent.h>
#include      <unistd.h>
#include      <stdlib.h>
#include      <string.h>

/**
 **   a simplified version of pwd
 **
 **   Starts in current directory and recursively
 **       climbs up to root of filesystem, prints top part
 **       then prints current part
 **   Uses readdir() to get info about each thing
 **/

void  get_info(char*, dev_t *, ino_t *);
void  printpathto(dev_t, ino_t);
void  inum_to_name(dev_t , ino_t , char *);

int main()
{
    ino_t  myinode ;
    dev_t  mydev   ;

    get_info(".", &mydev, &myinode);
    printpathto( mydev, myinode );           /* print path to here */
    putchar('\n');                          /* then add newline   */
    return 0;
}

void printpathto( dev_t this_dev, ino_t this_inode )
/*
 *   prints path leading down to an object with this dev,inode
 */
{
    ino_t  myinode , par_inode;
    dev_t  mydev,    par_dev;
    char   its_name[BUFSIZ];

    chdir( ".." );                          /* up one dir */

    inum_to_name(this_dev,this_inode,its_name ); /* get its name */

    get_info(".",&mydev,&myinode);           /* print head */
    get_info("..",&par_dev,&par_inode);
    if ( myinode != par_inode || mydev != par_dev )
        printpathto( mydev, myinode );      /* recursively */
    printf("%s", its_name );                /* now print */
                                           /* name of this */
}

/*
 * gets inode and dev for the specified path, uses lstat
 */
void get_info(char *path, dev_t *dp, ino_t *inp)
{
    struct stat info;

    if ( lstat(path, &info) == -1 ){
        perror(path);
        exit(2);
    }
    *inp = info.st_ino;
    *dp  = info.st_dev;
}

```

```
void inum_to_name(dev_t dev_to_find, ino_t inode_to_find , char *namebuf)
/*
 *   looks through current directory for a file with this inode
 *   number and copies its name into namebuf
 */
{
    DIR          *dir_ptr;           /* the directory */
    struct dirent *direntp;          /* each entry   */
    ino_t        cur_inum;
    dev_t        cur_dev;

    dir_ptr = opendir( "." );
    if ( dir_ptr == NULL ){
        fprintf(stderr, "cannot open a directory\n");
        exit(1);
    }
    /*
     * search directory for a file with specified inum and device
     */
    while ( ( direntp = readdir( dir_ptr ) ) != NULL ){
        get_info(direntp->d_name, &cur_dev, &cur_inum);
        if ( cur_inum == inode_to_find && cur_dev == dev_to_find )
        {
            strcpy( namebuf, direntp->d_name );
            closedir( dir_ptr );
            return ;
        }
    }
    fprintf(stderr, "error looking for inum %d\n", (int) inode_to_find);
    exit(1);
}
```