

# Operating System

## Assignment 02

Roll No.: CE092

**Aim:** Implementation of “pwd” and “ls” command in C. (use of getcwd, opendir, readdir, closedir)

## Various System Calls

### **getcwd, getwd, get current dir name: get current working directory**

**Synopsis:**        `#include <unistd.h>`

`char *getcwd(char *buf, size_t size);`

`char *getwd(char *buf);`

`char *get_current_dir_name(void);`

These functions return a null – terminated string containing an absolute pathname that is the current working directory of the calling process. The pathname is returned as the function result and via the argument buf, if present.

The getcwd() function copies an absolute pathname of the current working directory to the array pointed to by buf, which is of length size.

get\_current\_dir\_name() will malloc() an array big enough to hold the absolute pathname of the current working directory. If the environment variable PWD is set, and its value is correct, then that value will be returned, the caller should free the returned buffer.

getwd() does not malloc any memory. The buf argument should be a pointer to an array at least PATH\_MAX bytes long. If the length of the absolute pathname of the current working directory, including the terminating null byte, exceeds PATH\_MAX bytes, NULL is returned, and errno is set to ENAMTOOLONG.

### **opendir: open a directory**

**Synopsis:**        `#include <sys/types.h>`

`#include <dirent.h>`

`DIR *opendir(const char *name);`

The `opendir()` function opens a directory stream corresponding to the directory name, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

## **readdir: read a directory**

**Synopsis:**        `#include <dirent.h>`

`Struct dirent *readdir(DIR *dirp);`

The `readdir()` function returns a pointer to a `dirent` structure representing the next directory entry in the directory stream pointed to by `dirp`.

It returns `NULL` on reaching the end of the directory stream.

On Linux, the `dirent` structure is defined as follows:

```
struct dirent {
    ino_t      d_ino;           /* inode number */
    off_t      d_off;          /* offset to the next dirent */
    unsigned short d_reclen;    /* length of this record */
    unsigned char d_type;       /* type of file */
    char       d_name[256];     /* filename */
};
```

On success, `readdir()` returns a pointer to a `dirent` structure.

If the end of the directory stream is reached, `NULL` is returned and `errno` is not changed. If an error occurs, `NULL` is returned and `errno` is set appropriately.

## **closedir: close a directory**

**Synopsis:**        `#include <sys/types.h>`  
                  `#include <dirent.h>`

`int closedir (DIR *dirp);`

The `closedir()` function closes the directory stream associated with `dirp`.

A successful call to `closedir()` also closes the underlying file descriptor associated with `dirp`.

The directory stream descriptor `dirp` is not available after this call.

The `closedir()` function returns 0 on success.

On error, -1 is returned, and errno is set appropriately.

## Tasks

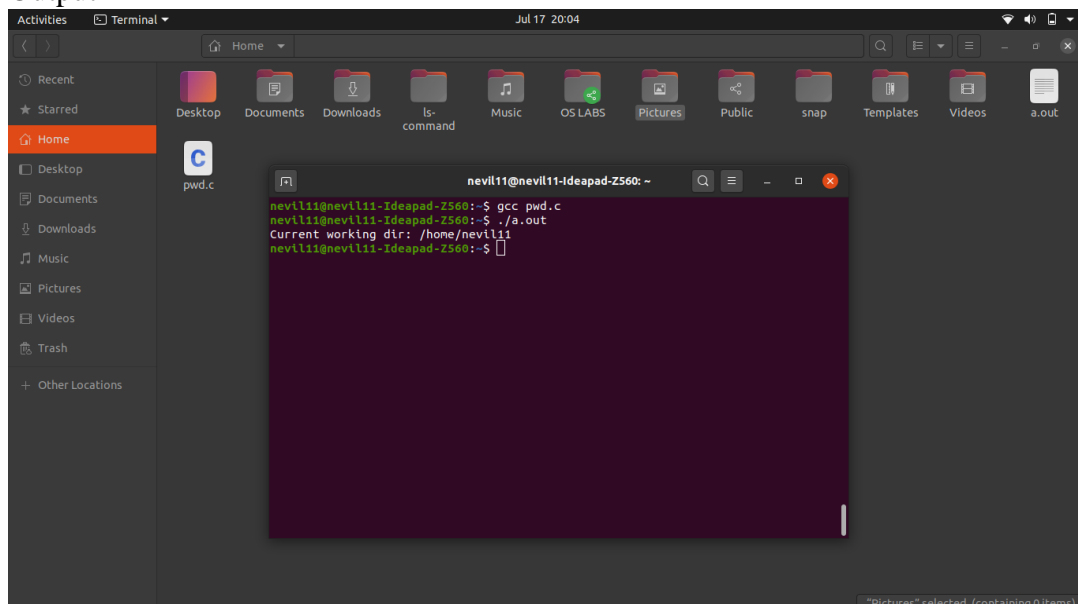
1. Write a program to get current working directory name of the current process. (“pwd” command).

- Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    char *run_dir = (char *)malloc(1024);
    if (getcwd(run_dir, 1024) != NULL)
    {
        fprintf(stdout, "Current working dir: %s\n", run_dir);
    }
    return 0;
}
```

- Output



2. Implement a program to list contents of current directory and extend it further to use “ls -r” command.

- Code

```

#include<stdlib.h>
#include<stdio.h>
#include<dirent.h>
#include<string.h>
void function(char *name)
{
    struct dirent *temp;
    DIR *t;
    if((t=opendir(name))==NULL)
        exit(0);
    strcat(name,"/");
    char n[256];
    strcat(n,name);
    while((temp=readdir(t)))
    {
        if((strcmp((temp->d_name),".")==0) || (strcmp((temp->d_name),"..")==0))
            continue;
        else
            printf("%s ",temp->d_name);
    }
    t=opendir(name);
    while((temp=readdir(t)))
    {
        if(strcmp((temp->d_name),".")==0)
        {
            continue;
        }
        else if(strcmp((temp->d_name),"..")==0)
        {
            continue;
        }
    }
}

```

```

        else
        {
            if((temp->d_type)==DT_DIR)
            {
                printf("%s",temp->d_name);
                printf("/");
                strcpy(n,name);
                strcat(n,temp->d_name);
                function(n);
                printf("\n");
            }
        }
    }
    printf("\n");
    closedir(t);
}

int main(int argc,int argv[])
{
    char buf[256],temp[256];
    struct dirent *dp;
    DIR *dirp;
    printf("Enter directory path:\n");
    scanf("%s",buf);
    if((dirp=opendir(buf))==NULL)
    {
        printf("Error");
        exit(1);
    }
    strcat(buf,"/");
    strcpy(temp,buf);

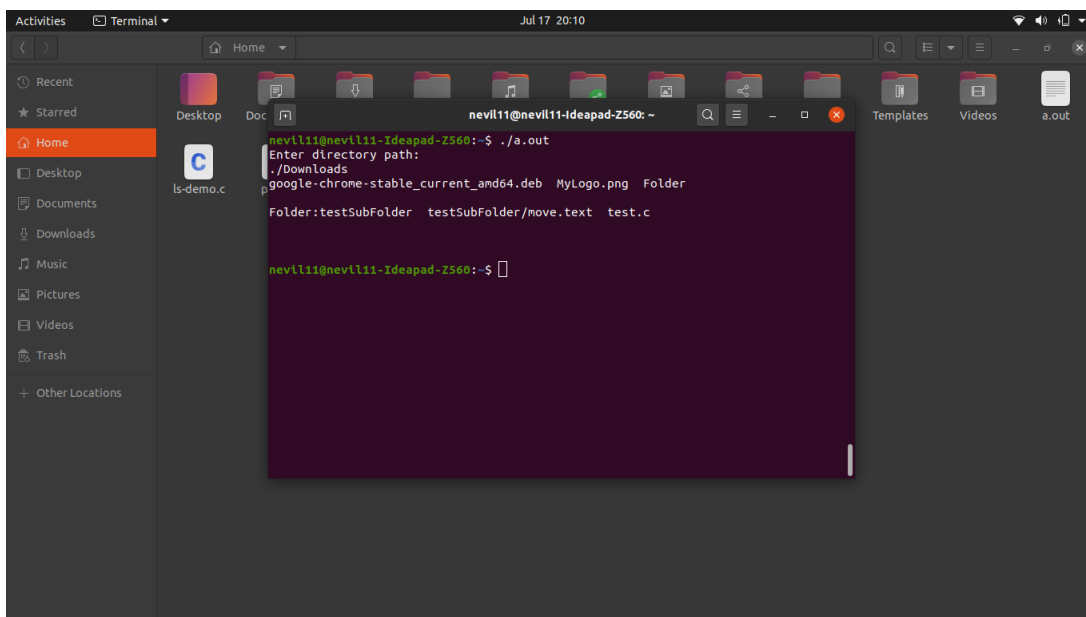
```

```

while((dp=readdir(dirp)))
{
    if((strcmp((dp->d_name),".")==0) || (strcmp((dp->d_name),"..")==0))
        continue;
    else
        printf("%s ",dp->d_name);
}
dirp=opendir(buf);
while((dp=readdir(dirp)))
{
    if((strcmp((dp->d_name),".")==0) || (strcmp((dp->d_name),"..")==0))
    {
        continue;
    }
    else
    {
        if((dp->d_type)==DT_DIR)
        {
            printf("%s",dp->d_name);
            printf(":");
            strcpy(temp,buf);
            strcat(temp,(dp->d_name));
            function(temp);
        }
    }
    printf("\n");
}
closedir(dirp);
}

```

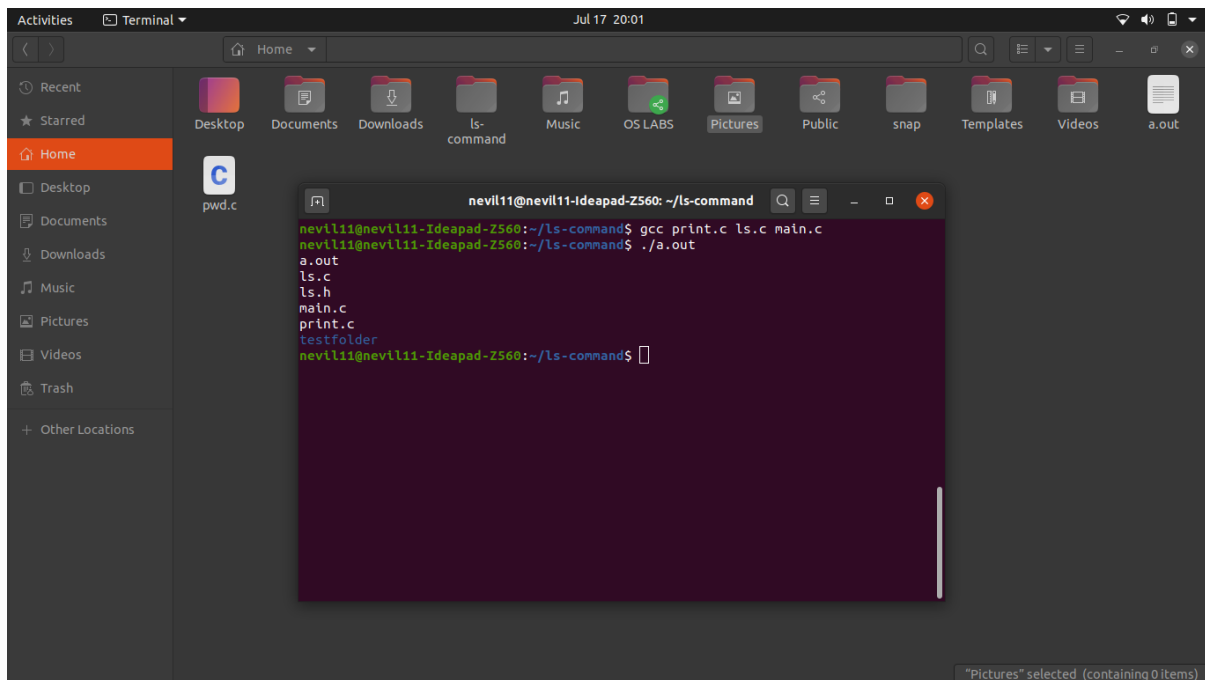
## - Output



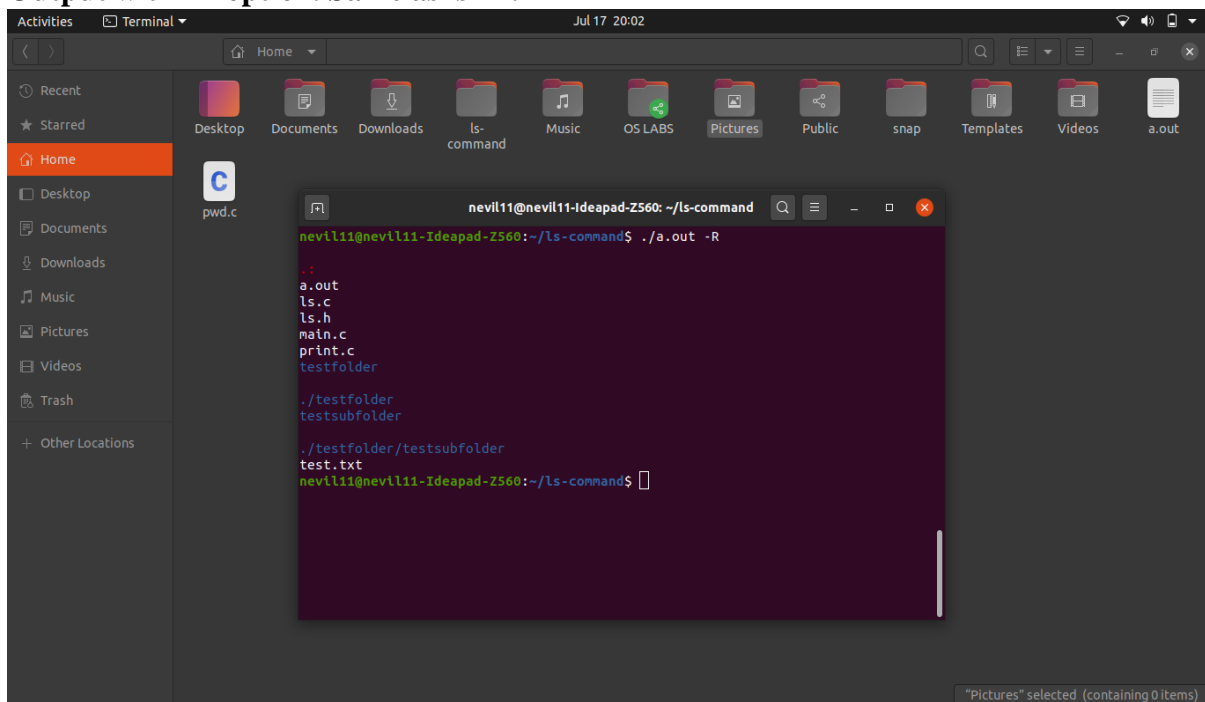
## EXTRA

- I have included one extra folder, which contains the proper implementation of ls command.
- It works with all the command line arguments possible with ls command in Linux.
- Please look at the instructions given in the HOW TO USE.txt file inside extra folder.
- Since the code is lengthy and command line argument options are more, to keep pdf concise and clean I will attach few of the screenshots in this document.
- The source code of the full ls command is available inside the extra folder in the zip file.

## Normal output. Same as ls.

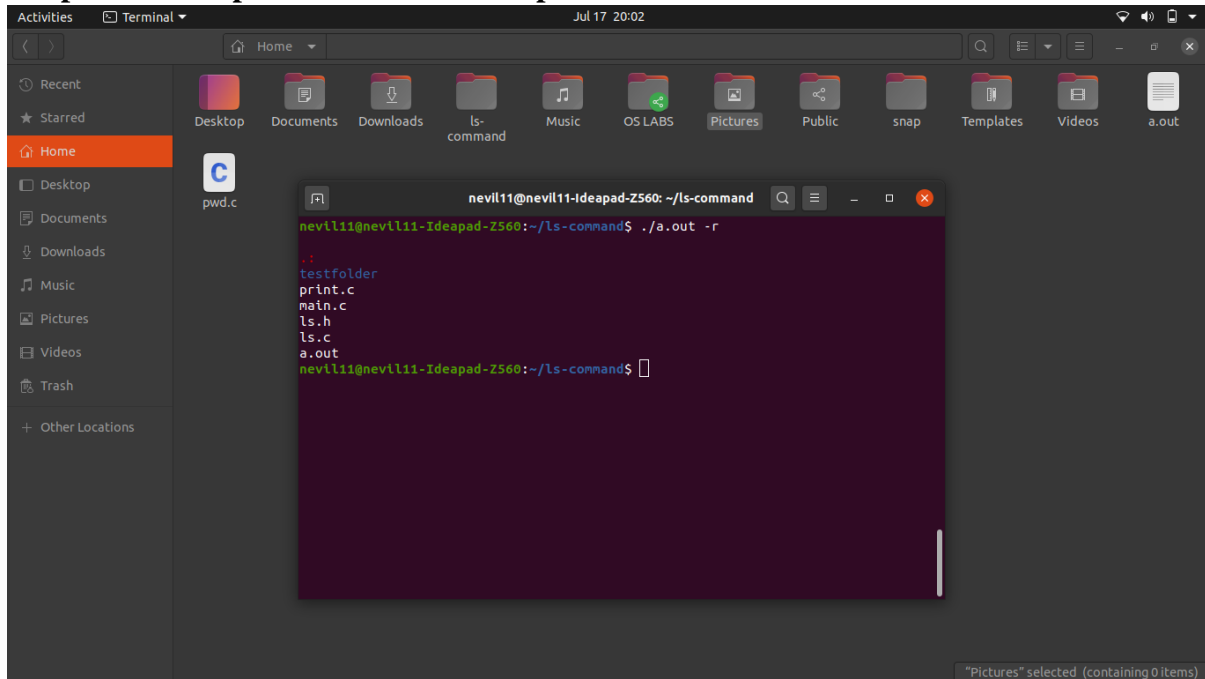


## Output with -R option. Same as ls -R.



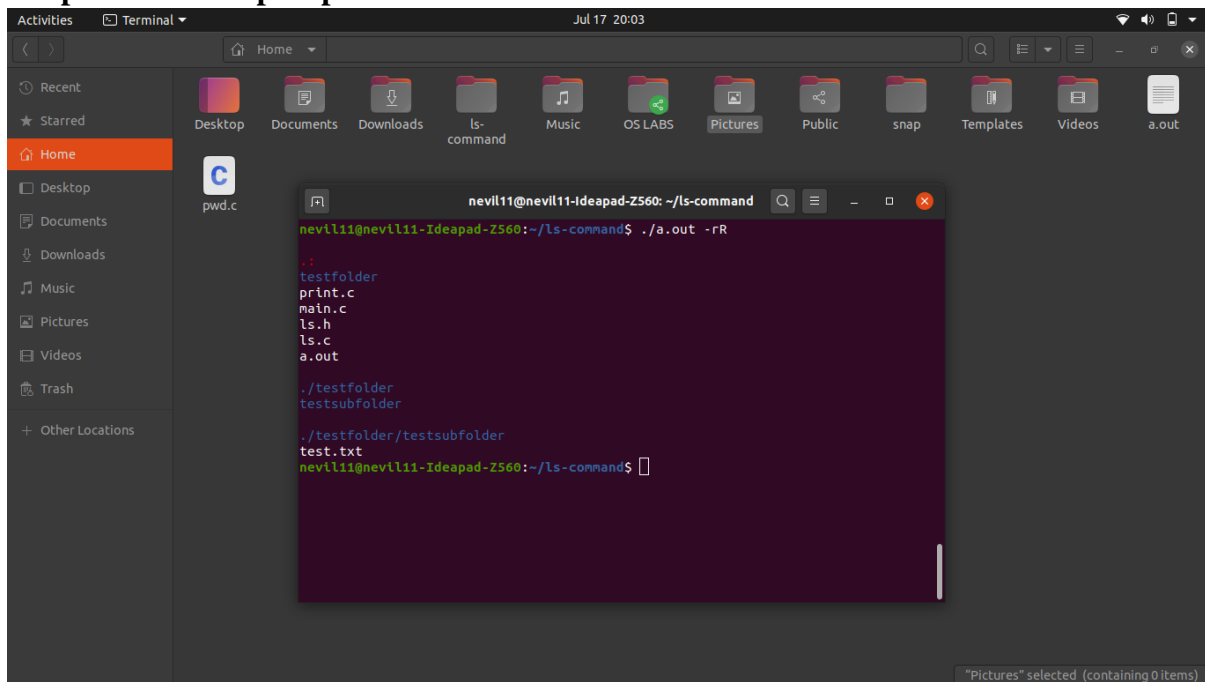


## Output with -r option. Same as ls -r to print in reverse order.



```
nevil11@nevil11-Ideapad-Z560: ~/ls-command
nevil11@nevil11-Ideapad-Z560:~/ls-command$ ./a.out -r
.:
testfolder
print.c
main.c
ls.h
ls.c
a.out
nevil11@nevil11-Ideapad-Z560:~/ls-command$
```

## Output with multiple options.

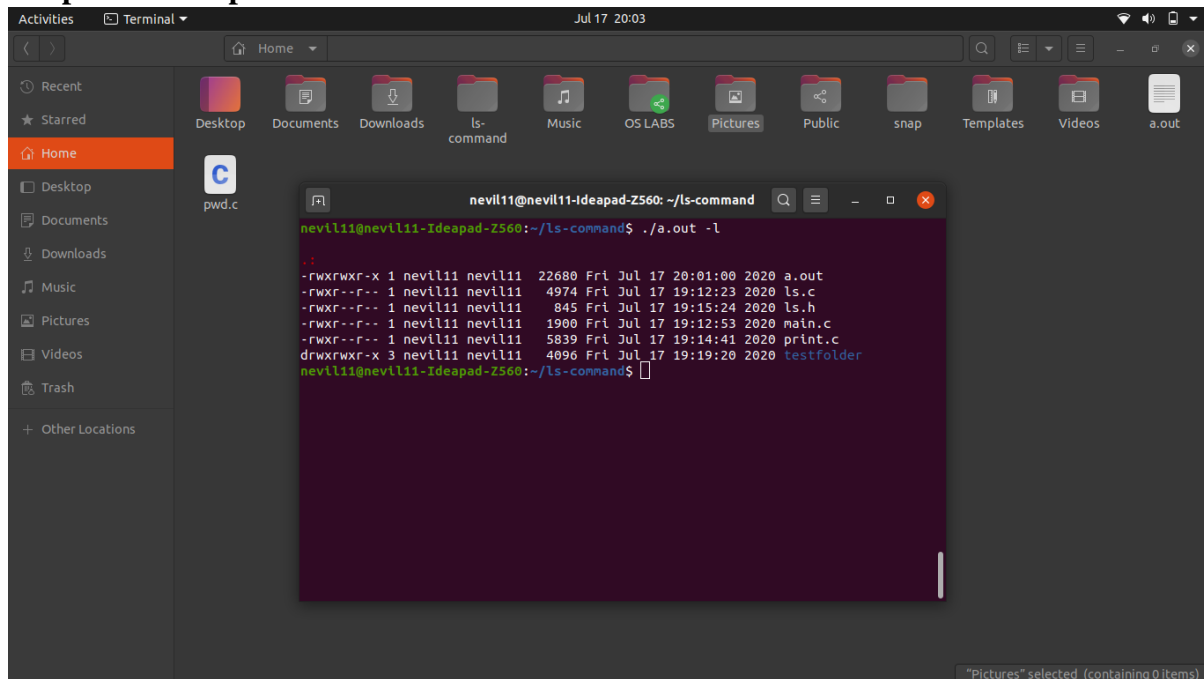


```
nevil11@nevil11-Ideapad-Z560: ~/ls-command
nevil11@nevil11-Ideapad-Z560:~/ls-command$ ./a.out -rR
.:
testfolder
print.c
main.c
ls.h
ls.c
a.out

./testfolder
testsubfolder

./testfolder/testsubfolder
test.txt
nevil11@nevil11-Ideapad-Z560:~/ls-command$
```

## Output with -l option. Same as ls -l.



## Output with argument to any folder with absolute / relative path.

