

LINUX FILE EXPLORER

Rong Xie and Shashank Gupta

Wednesday 5nd April

1 Introduction:

1.1 context

To manage a project of this stature, we needed to have simple knowledge of C programming, the Linux terminal environment, fundamentals of file browsing, .desktop files and, we did find resources online to aid us in some parts of the assignment and sources are listed below.

1.2 Problem Statement

We understand that it can be challenging as it was with us for the new Linux user to get instantly familiar with the Linux terminal commands especially when one is so used to the traditional Windows file browser using click. The issue we tackle in this project is that we make a new shell application (app) that we have attempted to provide a user-friendly and simple approach to those who aren't very familiar with Linux, along with that, we have hooked it up to the Linux visual file browser so the user can get a visual sense of his file browsing presence. We have attempted to provide better help in understanding the commands in the shell as well as how to use it by providing better error feedback. The added feedback is implemented for the users to provide them with an improved sense of directionality. What we hope to achieve with this project is to create a shell app that will help the new learners with its hopeful interactive abilities. What really motivates us to build on this project is that we have been so extremely overwhelmed by how much more we have gotten to learn about c programming from this project, this project has definitely broadened our horizons in such a short span of time.

1.3 Result

Our goal was to create a user-friendly shell app that aids its users by providing more feedback (text and visual). We were successfully able to hook Nautilus up with our program to provide the users with a visual sense as we browse through files, we also added a lot of print statements throughout the code to provide better feedback to the users. We worked on a total of 7 commands in our new shell (mkdir, rmdir, help, cd, exit, mv and copy). The desktop icon means that we no longer have to load up our program on the Linux terminal. A double click on the icon and our program starts running.

1.4 Outline

The rest of this report is structured as follows. Section 2 presents background information relevant to this project. Section 3 describes in detail the obtained result. The result is evaluated in Section 4. We conclude with Section 5.

2 Background Concepts

File browsers are a very crucial part of any computing experience. Even though browsing files using the command line is always an option, it can tend to be a little unfriendly/intimidating to the new users. This new file browser intends to help them out by providing a more interactive and visual approach.

We had no previous idea of coding with Gnome and many other libraries. Stephen Brennen has a brilliant article online that gave us an insight to as to how to create a basic shell. He created a basic UNIX shell and provided us with 3 function examples that are `cd`, `help` and `exit`. We have taken his work for an example and built much further on that to have it achieve our goals. Along with that a basic knowledge of `.desktop` files helped us achieve a desktop icon so we can run our program through a double click.

How to write a shell in c: Stephen Brennen wrote an article on a basic UNIX shell implementation. [1]

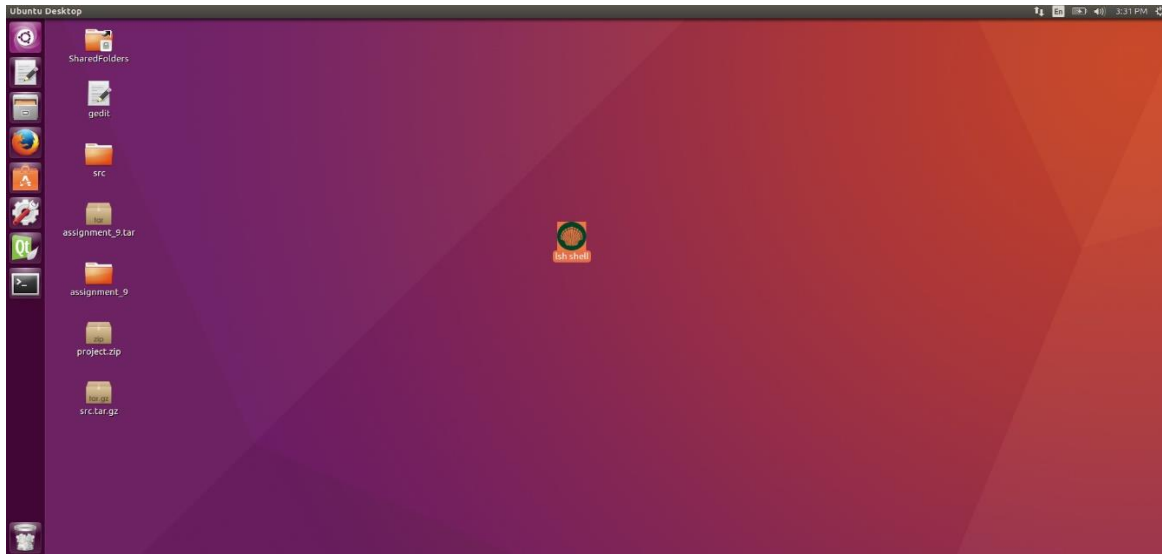
Desktop files: putting your application in the desktop menus: Desktop.gnome.org taught us how to make `.desktop` files. [2]

Ubuntu Manuals: Taught us how to open close Nautilus. [3]

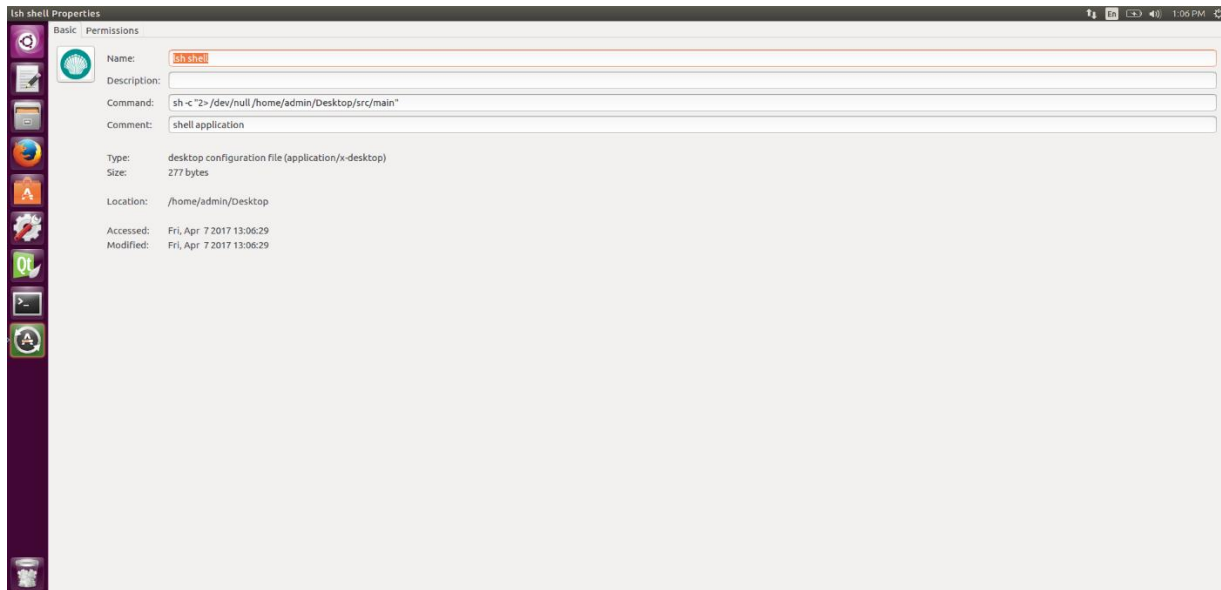
How to use `mkdir()` from `sys/stat.h`: Uli Kohler has an article on `mkdir` implementation and access types that we found useful. [4]

Section 3: Result

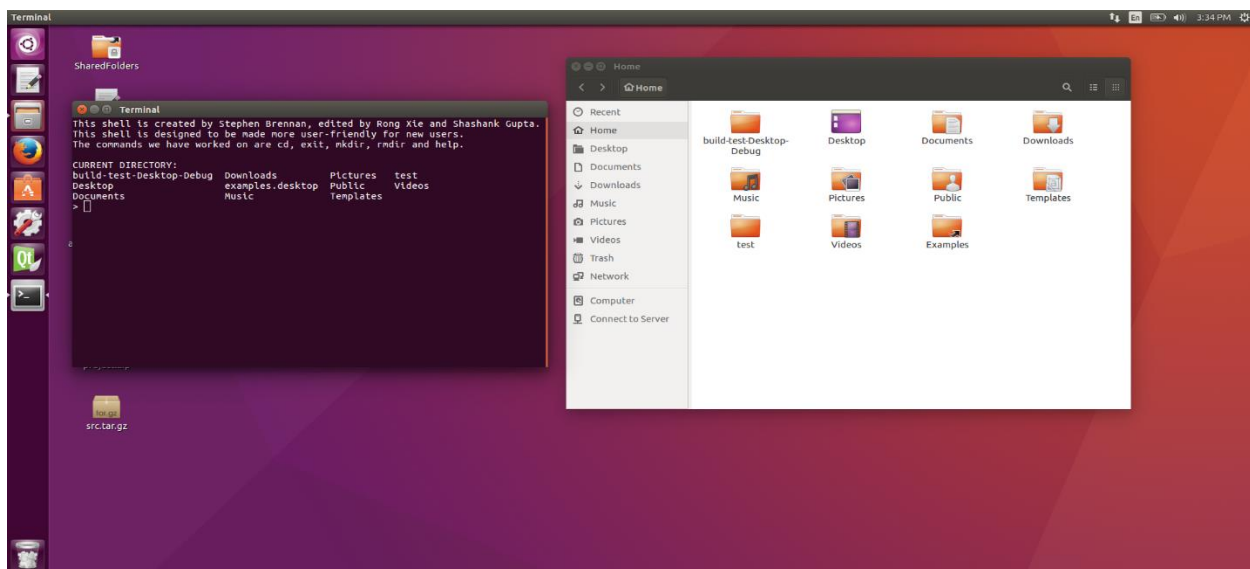
In this project, we have created a desktop app named “Ish shell”. “Ish shell” is a Linux command shell to execute the command that client entered. Compared to a regular Linux command shell, “Ish shell” provides an improved understanding of Linux commands as we produce more in detail feedback. The feedback would indicate the state of the execution and inform the client if the command has run successfully or not. If the command is not running successfully, our shell will then provide the user with an error with an example of how to execute the command e.g. Picture: 4.



Picture 1: the “Ish shell” icon in desktop. A double-click will run our program.



Picture 2: the app info of the app “Ish shell”. “2> /dev/null” sends all the stderr messages to dev/null. We couldn't not allow the console to print stderr messages as the nautilus file browser was printing a lot of terminal message that messed with the look of the program. Keeping our own error messages kept the program neat and flowing.

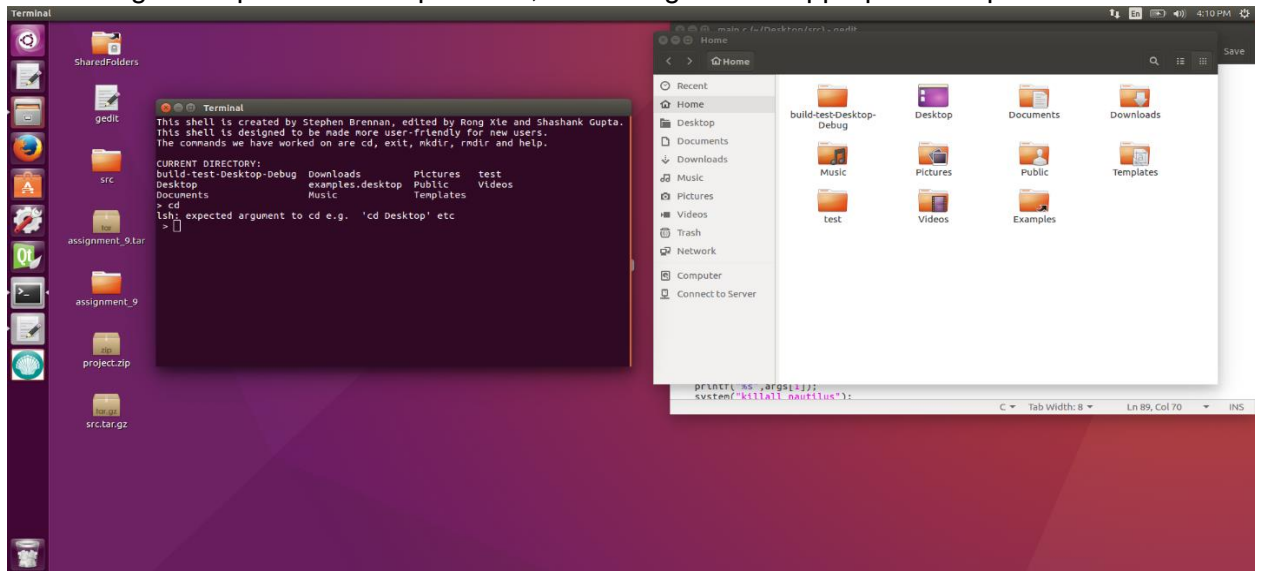


Picture 3: (“Ish shell”) when the user double-clicks the icon of “Ish shell”, the shell we created runs. On the left is the terminal window of the shell, it lets the user enter the commands. On the right is Nautilus, the traditional Linux file browser. It gives the user a visual sense of his file browsing presence.

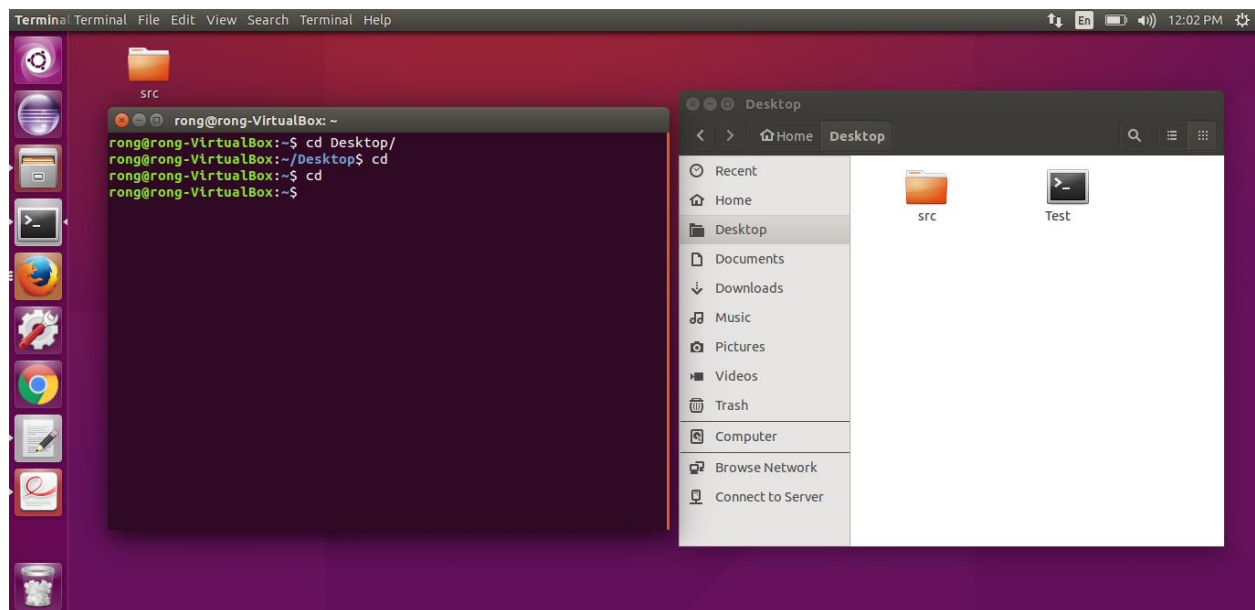
This project is an “Ish shell” app with 7 implemented commands (change dir, make dir, remove dir, copy, remove, help, quit).

- The command “cd” would change the current directory to another existed directory in the “Ish shell” terminal and the loaded graphic user interface. If the directory does not exist

or if an argument past cd is not provided, the shell gives the appropriate response.



Picture 4: (above) ("lsh shell") when the cd entered without a preceding argument.



Picture 5: (regular shell) when we test the cd command in the same fashion on the regular terminal, we can see that it provides no feedback to this approach.

```

main.c (-/Desktop/src) - gedit
/*
 * Bulltin function implementations.
 */
/**
 * @brief Bulltin command: change directory.
 * @param args List of args. args[0] is "cd". args[1] is the directory.
 * @return Always returns 1, to continue executing.
 */
int lsh_cd(char **args)
{
    // size of command
    char command[50];
    char *cwd;
    char buff[PATH_MAX + 1];

    if (args[1] == NULL) {
        printf("lsh: expected argument to cd e.g. 'cd Desktop' etc\n"); // serves as an example
        free(cwd); // frees the pointer we created previously
        return 1; // returns to our shell
    } else if (strcmp(args[1], "..") == 0) { // checks if argv[1] == ".." using strcmp
        system("killall nautilus"); // closes nautilus
        chdir(".."); // changes directory, goes a directory back
    }

    // checks if the directory mentioned exists or not
    else if (chdir(args[1]) != 0) {
        printf("lsh error: directory does not exist!\n"); // prints error if directory aforementioned exists or not.
    }

    cwd = get_current_dir_name(); // gets current directory name
    if (cwd != NULL) { // checks if current directory exists
        system("killall nautilus"); // kills nautilus before it restarts it!
        printf("my working directory is: %s\n", cwd); // prints the message before command line argument
        sprintf(command, "gnome-open %s", cwd); // give command string a value using sprintf.
        printf("%s", cwd); // prints the directory path right before the argument '>'
        system(command); // executes the value that we passed on to the command string
    }
    free(cwd); // frees the pointer
    return 1; // return to our shell
}

int lsh_cp(char **argv)
{
    char buffer[1024];
}

```

Picture 6: cd command's function commented in detail.

We modified the very basic original cd function. Here is where a lot of Nautilus restarting happens using the system gnome – open call. Each time we enter a new directory, we shut nautilus and restart it at the desired directory. We allocate a pointer 'cwd' in this function that we also free. Upon valgrinding this program we detected no memory loss.

```

main.c (-/Desktop/src) - gedit
/*
 * void lsh_loop(void)
 * {
 *     char *line;
 *     char **args;
 *     int status;
 *
 *     chdir("..");
 *     chdir("..");
 *     printf("\a"); // clears screen for the user
 *     printf("This shell is created by Stephen Brennan, edited by Rong Xie and Shashank Gupta.\n");
 *     printf("This shell is designed to be made more user-friendly for new users.\n");
 *     printf("The commands we have worked on are cd, exit, mkdir, rmdir and help.\n");
 *     printf("CURRENT DIRECTORY:\n");
 *     system("ls"); // prints the contents to the current directory
 *     system("nautilus --browser"); // opens the nautilus browser to home
 *     do {
 *         printf("> ");
 *         line = lsh_read_line();
 *         args = lsh_split_line(line);
 *         status = lsh_execute(args);
 *
 *         free(line);
 *         free(args);
 *     } while (status);
 * }
 *
 * bool isDirectoryEmpty(char *dirname) {
 *     int n = 0;
 *     struct dirent *d;
 *     DIR *dir = opendir(dirname);
 *     if (dir == NULL) // Not a directory or doesn't exist
 *         return 1;
 *     while ((d = readdir(dir)) != NULL) {
 *         if (++n > 2)
 *             break;
 *     }
 *     closedir(dir);
 *     if (n <= 2) // Directory Empty
 *         return true;
 *     else
 *         return false;
 * }
 *
 * /**
 * @brief Main entry point.
 * @param argc Argument count.
 */

```

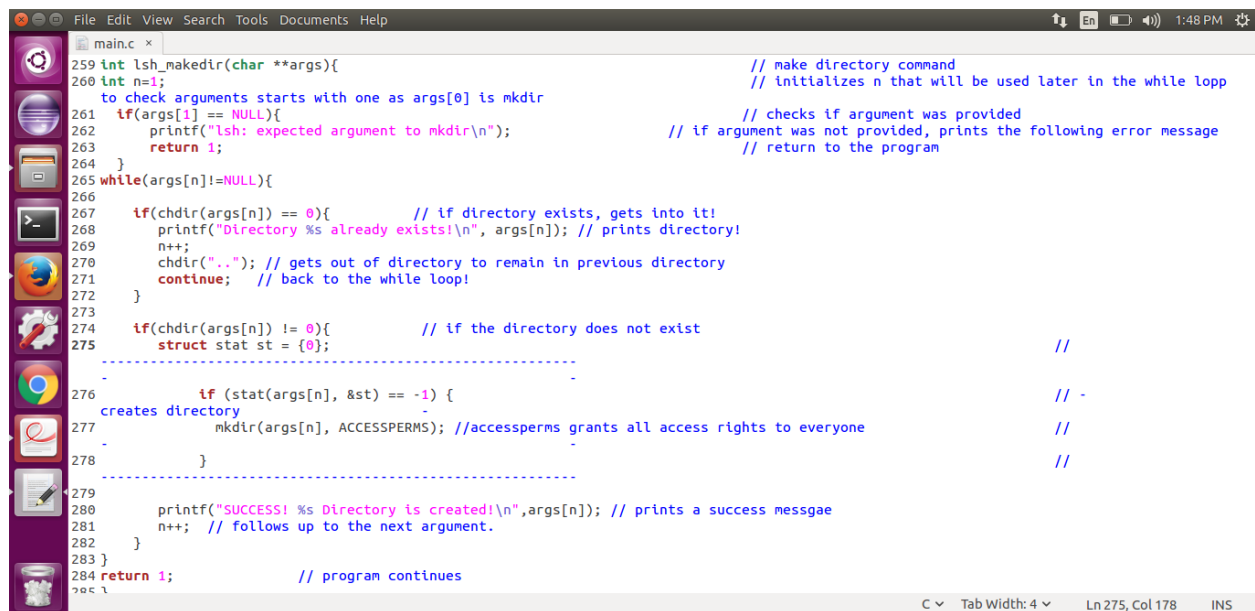
Picture 7: loop function and the isDirectoryEmpty function

The loop function was provided by Stephen Brennen in the original code but we have made very minor tweaks to it such as:

- 1) The moment we start the program, the current directory is home.
- 2) It print the contents of the current directory.
- 3) Displays current directory in Nautilus

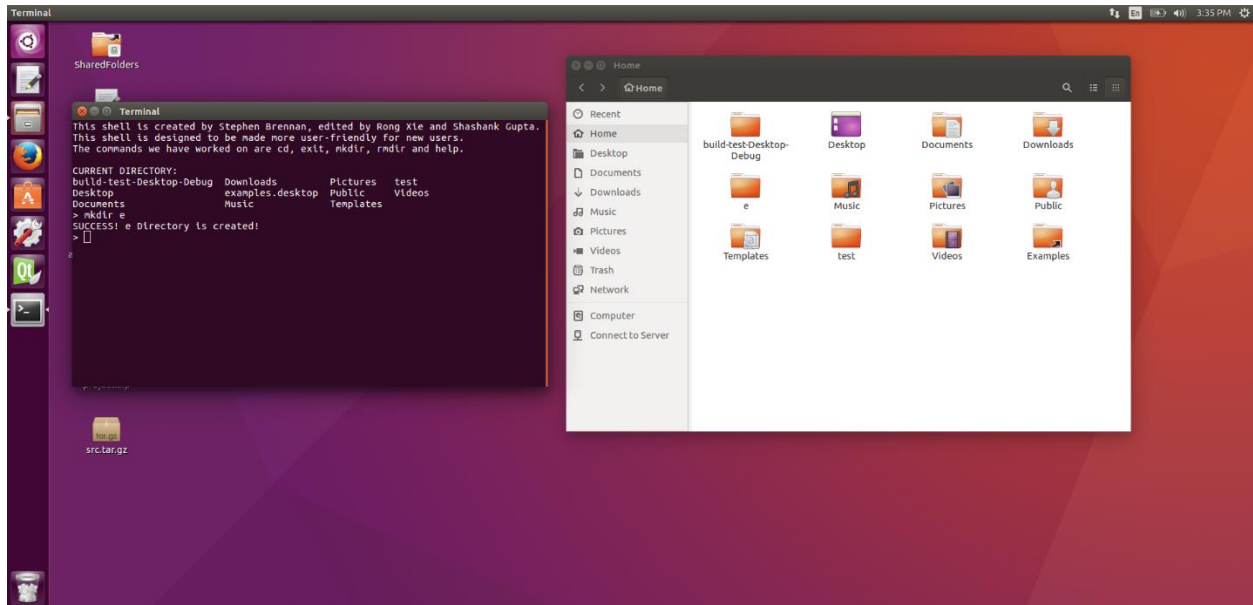
The isDirectoryEmpty function checks if a directory is empty. We use this function in the rmdir function to check if the directory we want to remove has contents or not.

- The command “mkdir” creates a new directory. If a preceding argument is not entered, the directory already exists or, the command runs successful, our shell responds promptly.

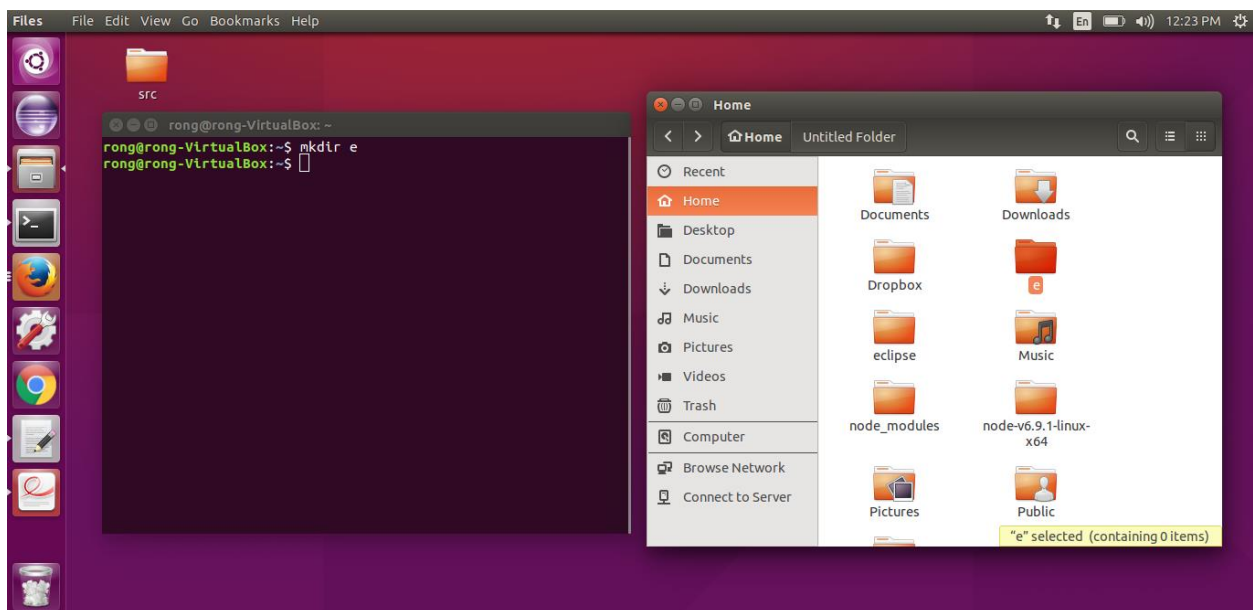


```
259 int lsh_mkdir(char **args){
260     int n=1;
261     // make directory command
262     // initializes n that will be used later in the while loop
263     to check arguments starts with one as args[0] is mkdir
264     if(args[1] == NULL){
265         printf("lsh: expected argument to mkdir\n");
266         // checks if argument was provided
267         // if argument was not provided, prints the following error message
268         return 1;
269         // return to the program
270     }
271     while(args[n]!=NULL){
272         if(chdir(args[n]) == 0){
273             // if directory exists, gets into it!
274             printf("Directory %s already exists!\n", args[n]); // prints directory!
275             n++;
276             chdir(".."); // gets out of directory to remain in previous directory
277             continue; // back to the while loop!
278         }
279         if(chdir(args[n]) != 0){
280             // if the directory does not exist
281             struct stat st = {0};
282             //
283             if (stat(args[n], &st) == -1) {
284                 // -
285                 creates directory
286                 mkdir(args[n], ACCESSPERMS); //accessperms grants all access rights to everyone
287                 //
288             }
289             //
290             printf("SUCCESS! %s Directory is created!\n",args[n]); // prints a success msggae
291             n++; // follows up to the next argument.
292         }
293     }
294     return 1;
295     // program continues
296 }
```

Picture 8: code of mkdir function implemented in “lsh shell” program with detailed comments of execution. Like the original terminal, we made a while loop so we can make multiple directories at once.



Picture 9: In our shell we entered the command “mkdir e” and, the shell responds with “success! A Directory is created!” to tell the user that the command ran successfully. As the command responds with success, Nautilus promptly displays the new directory.

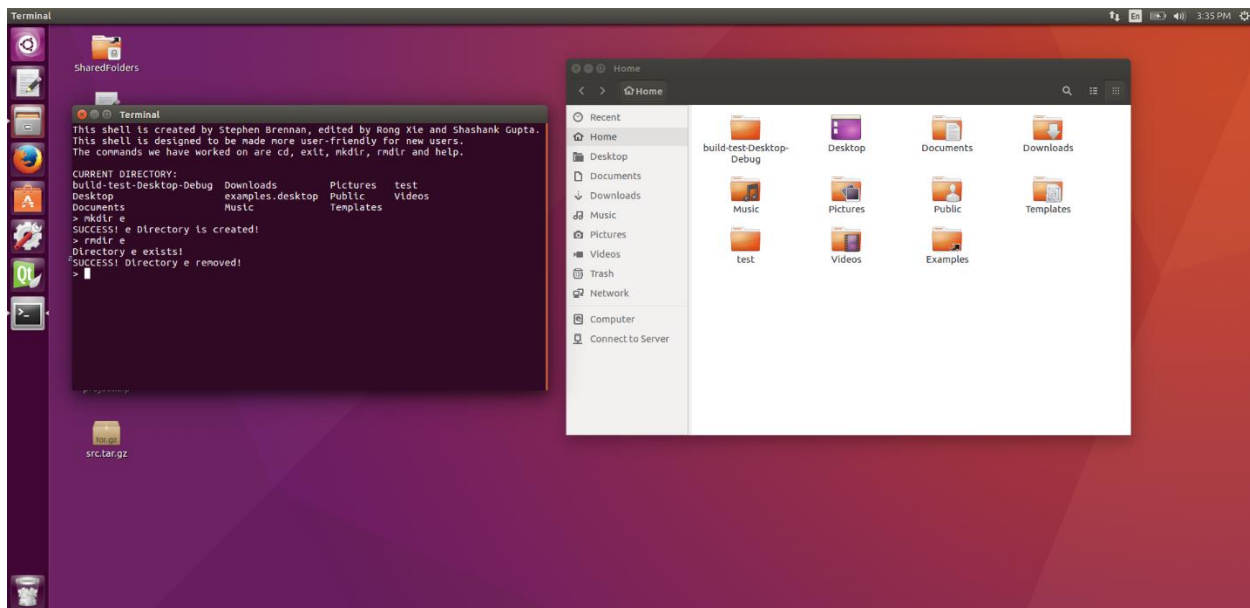


Picture 10: (regular shell) mkdir e in the regular terminal.

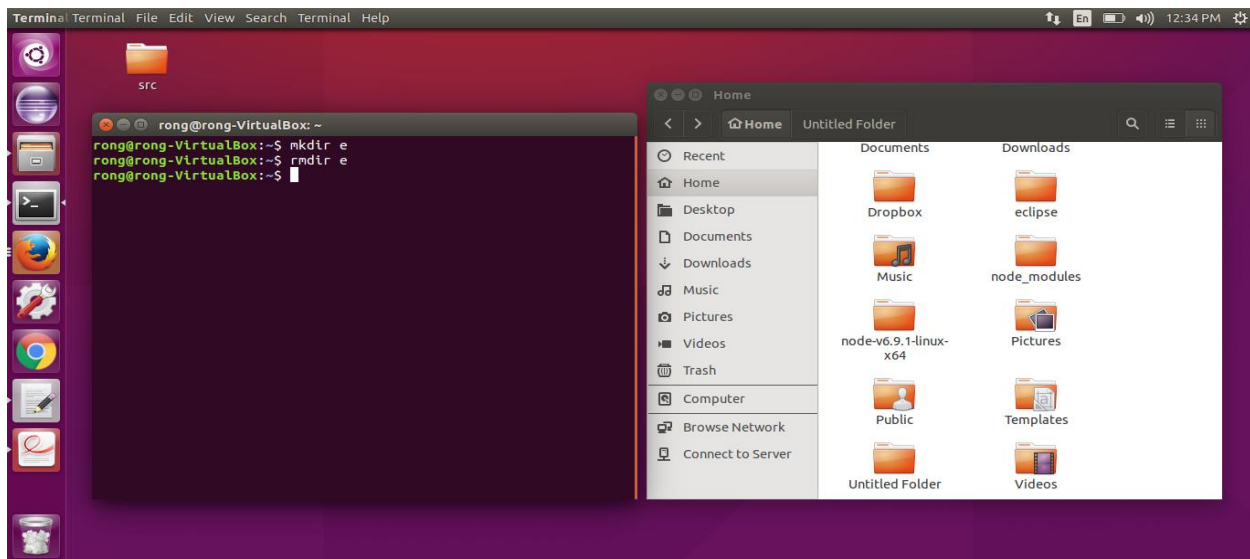
- The command “rmdir” removes an existing directory provided it is empty. If a preceding argument is not entered, the directory does not exist or, the command runs successful, the shell follows up promptly.


```
File Edit View Search Tools Documents Help
main.c x
223 int lsh_rmdir(char **args){ // remove directory command
224     int n=1; // initializes n that will be used later in the while loop
    to check arguments starts with one as args[0] is mkdir
225
226     if(args[1] == NULL){ // checks if argument was provided
227         printf("lsh: expected argument to rmdir e.g. rmdir a\n"); // if argument was not provided, prints the following error message
228         return 1; // return to the program
229     }
230
231     while(args[n]!=NULL){
232
233         if(isDirectoryEmpty(args[n])==false){ // checks if the directory we wish to remove is empty or not
234             printf("Directory %s exists!\n", args[n]); // prints the following message.
235             return 1; // returns to shell, program continues
236         }
237
238         if(chdir(args[n]) == 0 && isDirectoryEmpty(args[n]) == true){ // if directory exists, gets into it!
239             printf("Directory %s exists!\n", args[n]); // prints directory
240             printf("SUCCESS! Directory %s removed!\n", args[n]); // prints that the requested directory has been removed
241             chdir(".."); // gets out of directory to remain in previous directory
242             rmdir(args[n]); // removes the directory
243             n++; // checks other arguments so we can remove multiple directories at once
244             continue; // back to the while loop!
245         }
246         if(chdir(args[n]) != 0){ // if the directory does not exist
247             printf(" %s Directory does not exist!\n", args[n]); // prints the following message
248             n++; // goes to the next following request
249         }
250     }
251     return 1; // program continues
252 }
253 }
```

Picture 11: the code of rmdir function implemented in “lsh shell” program with detailed comments.

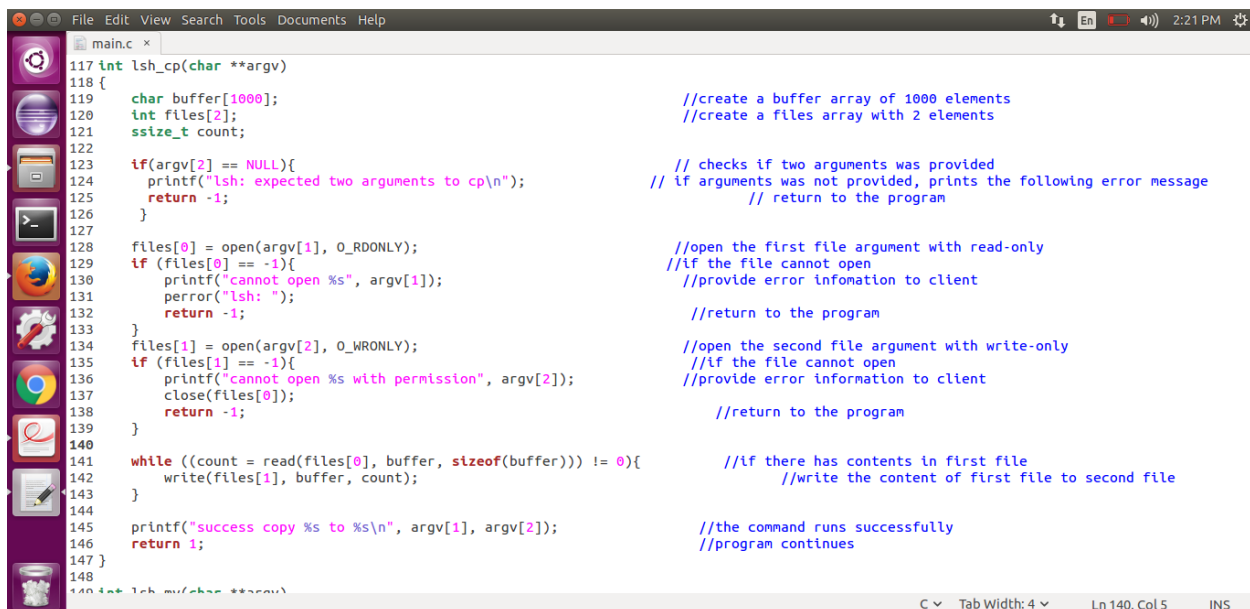


Picture 12: (“lsh shell”) we enter command “rmdir e” in the terminal shell, and the shell responds with “success! e Directory is removed!” to tell user that the command ran successfully. Nautilus too promptly discards that directory.

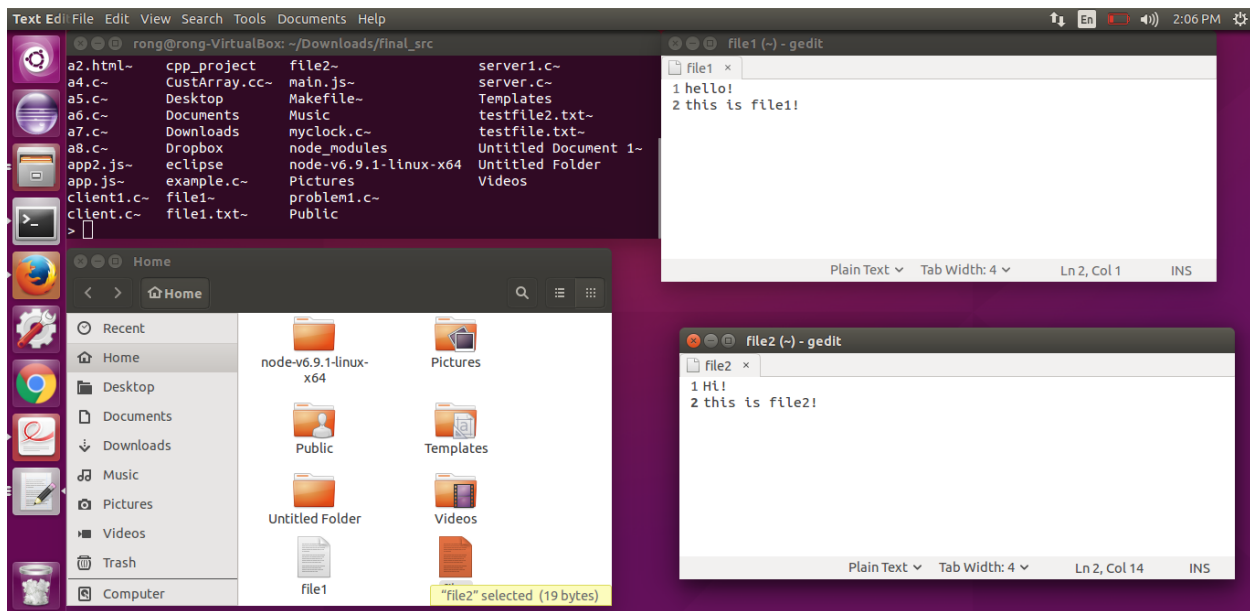


Picture 13: (regular shell) we enter the same command “rmdir e”.

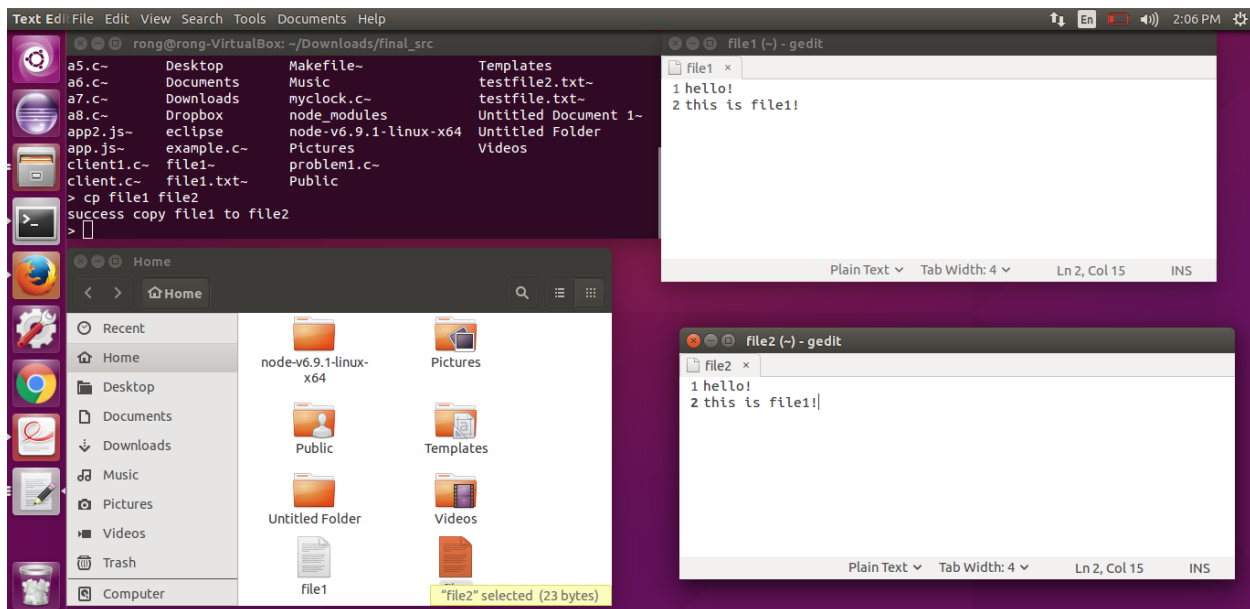
- The command “cp” copies the file argument1 to file argument 2. If enough arguments are not provided, the permissions of two file are not correct, two files do not exist or the command runs successful, the shell responds promptly.



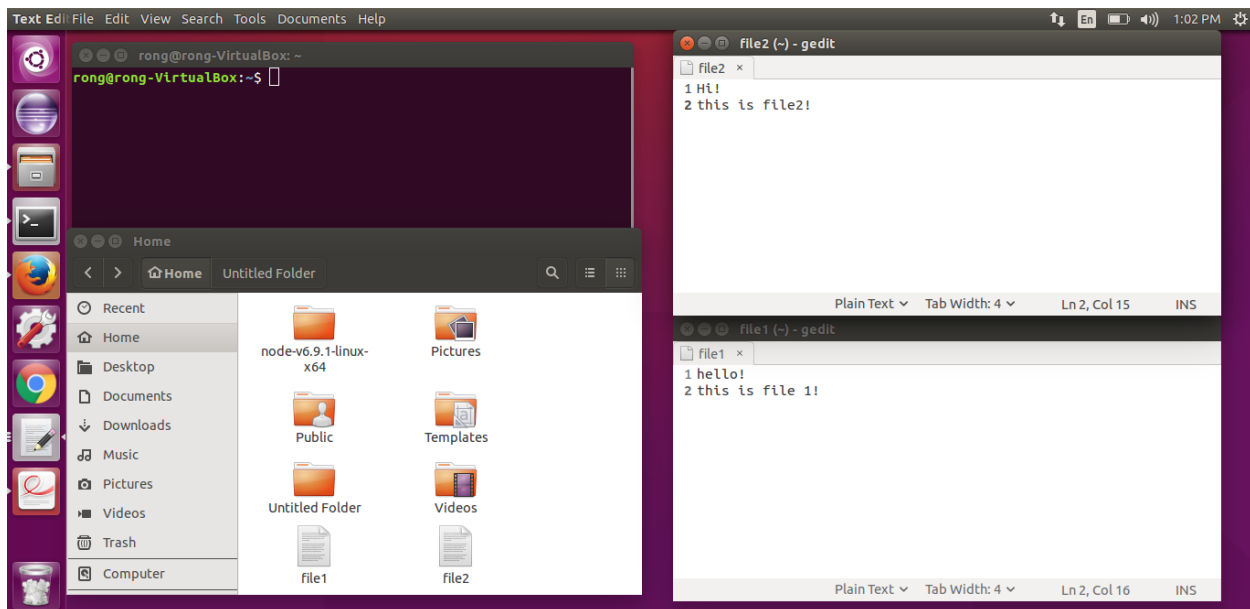
Picture 14: the code of cp command implements in “lsh shell” with detail comments



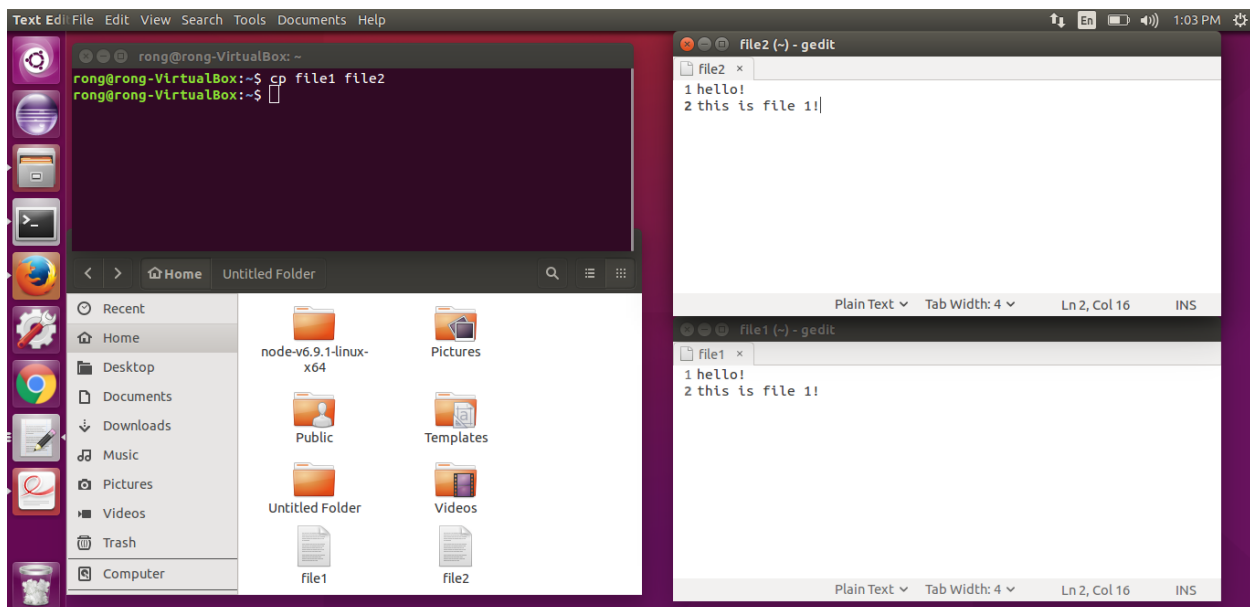
Picture15 :("Ish shell") we have two files in the current directory named "file1" and "file2". And they have different contents.



Picture 16: ("Ish shell") when we run the command "cp file1 file2", the content of file1 would copy to file2 or overwrite in file2. After successful running, the shell would provide feedback of "success copy file1 to file2" to indicate the running state.



Picture 17: (regular shell) we have two files in the current directory named “file1” and “file2”. And they have different contents.



Picture 18: (regular shell) when we run the same command “cp file1 file2” in regular shell, the content of file1 would copy to file2 or overwrite in file2 without feedback.

- The command “mv” removes the file argument1 and creates a backup file argument 2. If enough arguments are not provided, the permissions of two file are not correct, two files do not exist or the command runs successful, the shell responds promptly.

```

149 int lsh_mv(char **argv){
150     struct stat buf_file1, buf_file2;
151     char *file1, *file2, *new_file1, *new_file2;
152     char *root_directory;
153
154     if(argv[2] == NULL){
155         printf("lsh: expected argument to mv\n");
156         return -1;
157     }
158
159     file1 = (char*)malloc(strlen(argv[1]) + 1);
160     file2 = (char*)malloc(strlen(argv[2]) + 1);
161     strcpy(file1, argv[1]);
162     strcpy(file2, argv[2]);
163
164     stat(file1, &buf_file1);
165     stat(file2, &buf_file2);
166
167     printf("%s is a ", file1);
168     if (S_ISREG(buf_file1.st_mode)) {
169         puts("a regular file");
170     }
171     if (S_ISDIR(buf_file1.st_mode)) {
172         puts("a directory");
173     }
174
175     printf("%s is a ", file2);
176     if (S_ISREG(buf_file2.st_mode)) {
177         puts("a regular file");
178     }
179     if (S_ISDIR(buf_file2.st_mode)) {
180         puts("a directory");
181     }
182 }

```

Comments in the code explain the purpose of each step: creating stat structures, allocating memory for file names, copying arguments, and checking file types (regular file or directory).

Picture 19: The code of cp command's function implemented in "lsh shell" with detailed comments

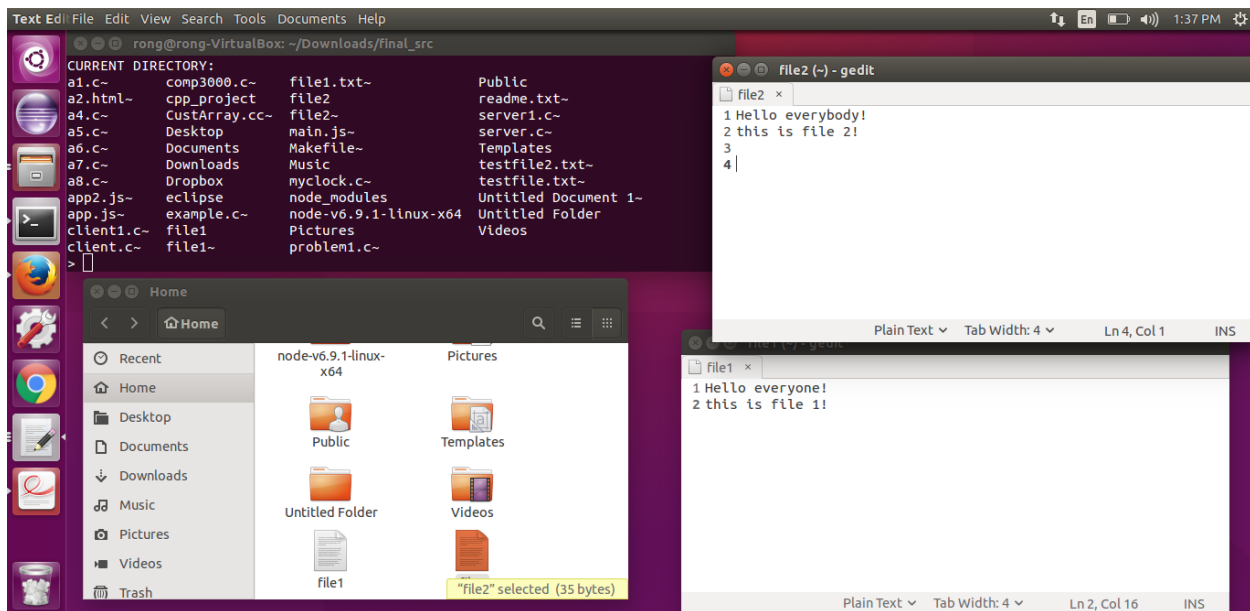
```

183 if (SAME_INODE(buf_file2, buf_file1)) {
184     printf("%s and %s are the identical\n", src, dest);
185     return -1;
186 }
187
188 root_directory = getenv("HOME");
189 printf("root directory is %s\n", root_directory);
190
191 new_src = (char*) malloc(strlen(file1) + 1 + strlen(root_directory) + 1);
192 strcpy(new_file1, root_directory);
193 strcat(new_file1, file1);
194 printf("new_src = %s\n", new_file1);
195
196 new_dest = (char*) malloc(strlen(file2) + 1 + strlen(root_directory) + 1);
197 strcpy(new_file2, root_directory);
198 strcat(new_file2, file2);
199 printf("new_dest = %s\n", new_file2);
200
201 if(rename(new_file1, new_file2) != 0){
202     fprintf(stderr, "rename failed with error %s\n", strerror(errno));
203     return -1;
204 }
205
206 free(new_file1);
207 free(new_file2);
208 free(file1);
209 free(file2);
210 return 1;
211 }

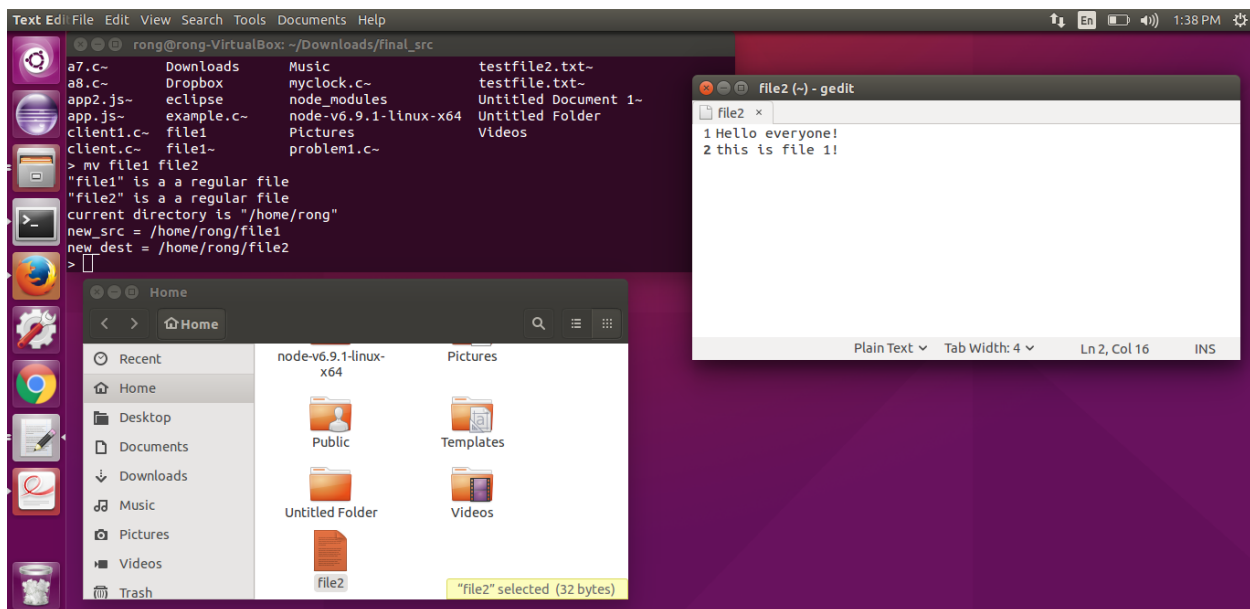
```

Comments in the code explain the purpose of each step: checking if files are identical, getting the root directory, allocating memory for new file paths, concatenating the directory and file names, and finally renaming the file and freeing memory.

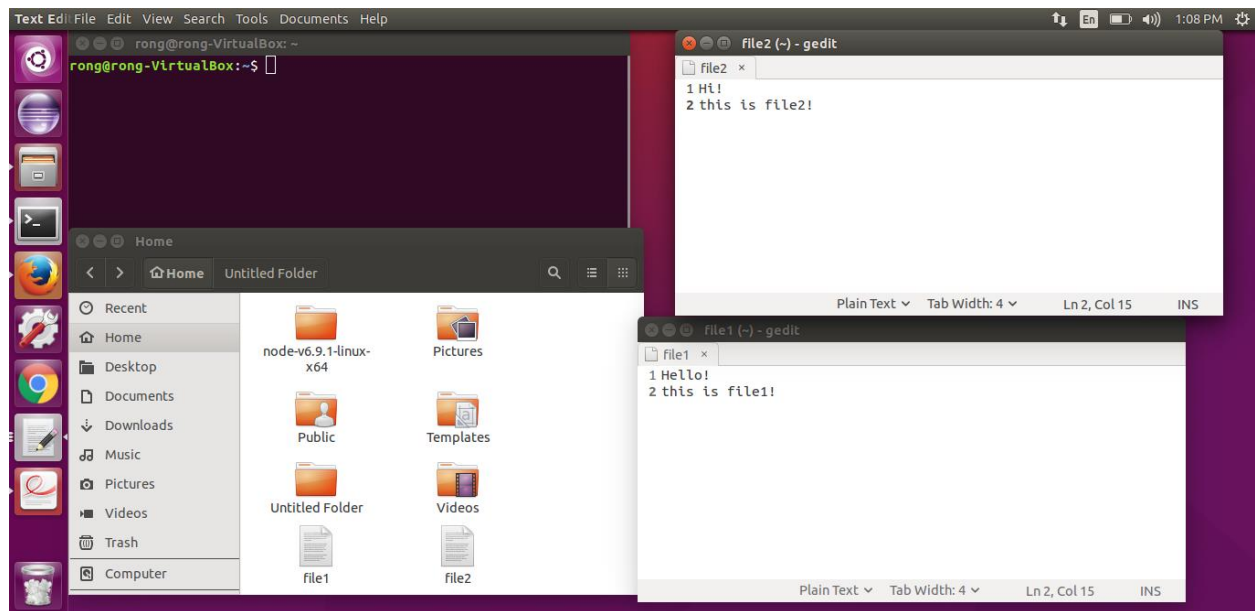
Picture 20: The code of cp command implemented in "lsh shell" with detailed comments (continued)



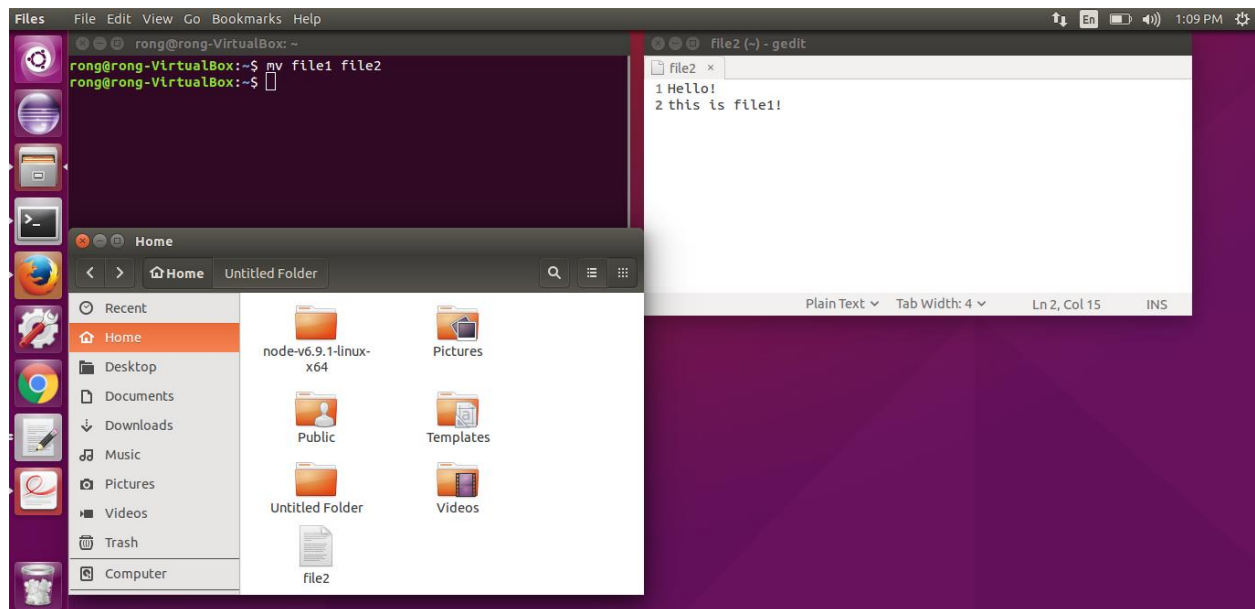
Picture 21:("Ish shell") we have two files in the root directory named "file1" and "file2". And they have different contents.



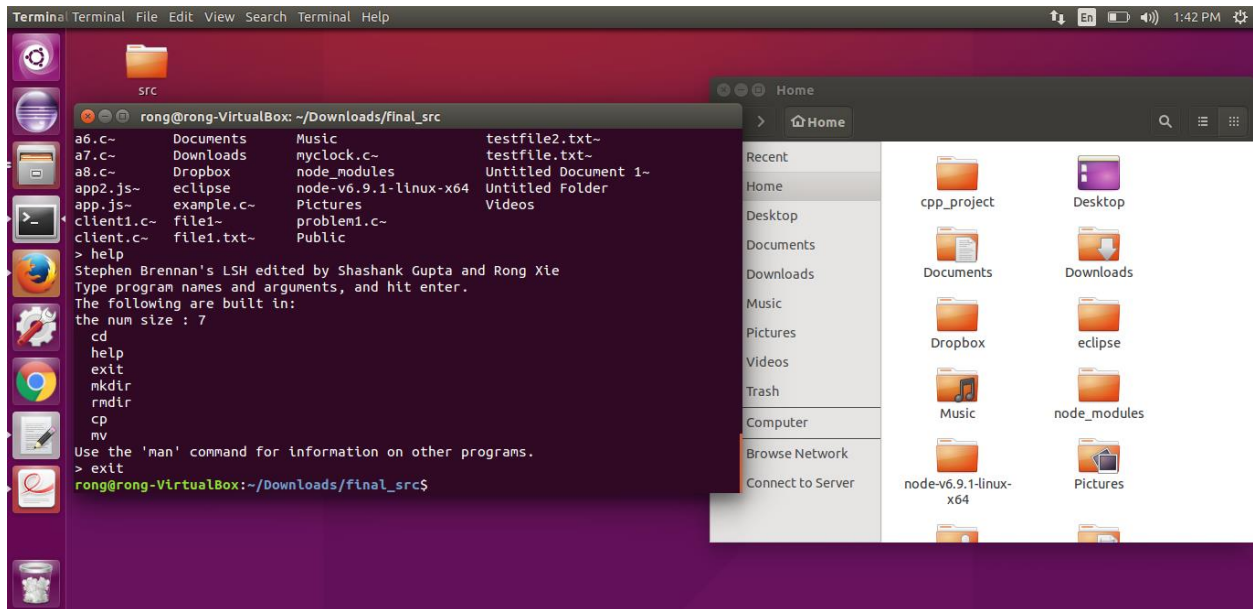
Picture 22: ("Ish shell") when we run the command "mv file1 file2", the content of file1 would copy to file2 or overwrite in file2, at the same time we delete the file1. After successfully running, the shell would provide appropriate feedback.



Picture 23: (regular shell) we have two files in the root directory named “file1” and “file2”. And they have different contents.



Picture 24: (regular shell) when we run the same command “mv file1 file2” in regular shell, the content of file1 would overwrite to file2, at the same time delete file1 without providing any sort of response.



Picture 25: ("lsh shell") the execution of "help" and "exit" command in "lsh shell".

Our idea is to provide new users with a more interactive display so they can get a better understanding of the terminal. This project is meant to apply to the beginners unfamiliar with the Linux terminal. We hope this project benefits those who are trying to learn the Linux terminal.

Section 4: Evaluation

Our goal was to make a user-friendly shell as aforementioned multiple times. We conducted usability tests to judge our performances. We tested very different kinds of people and, the methodologies and results are provided as follows:-

We asked our program testers to browse to a particular destination in the file browser along with some other mkdir rmdir commands.

- 1) Sammy Gentile- Never used Linux or a terminal, we provided him the basic general shell and our shell. We first gave the normal terminal and then our program. We asked him to reach a particular destination in the file browser. Upon using both, he stated that he preferred our program since it starts up listing a few commands that he used 'man' command to read over and found Nautilus very helpful to keep track of his presence.

- 2) Joel Kaminski- an ex-computer science student. Upon asking him to do the same thing as Sammy Gentile, he quickly figured out a way to the destination, he also carried out the cp, mkdir, rmdir commands. He stated that he prefers using the normal terminal but that's only since he is familiar with Linux; however, for a new user, he stated that he would definitely recommend our shell.
- 3) Manish Saraf- A current computer science student in his 3rd year had a similar opinion to Joel Kaminski. Although impressed by the addition of Nautilus and the app feature stated that he would prefer using the normal terminal as Nautilus takes up unnecessary space on the monitor for someone who is familiar with his browsing presence. He however prefers the better feedback that we provide. Like Joel, he too stated that our program will definitely be more suited towards the new learners as they would prefer a more interactive application.
- 4) Sarah – Another computer science student in her 3rd year stated that she loved the idea of pairing Nautilus with the shell however would prefer more monitor space like Manish. She too liked the added comments and stated that the program is better suited for beginners e.g. 1st years.

Our usability test went just as we predicted. New beginners like Sam preferred our program as contrary to the more advanced users.

Section 5: Conclusion

5.1 Summary

To summarize our work, we created a new user-friendly shell application that is hooked up with Nautilus to give a visual sense as we use the terminal. We have the hope that this program aids the new users unfamiliar with terminal. We have worked on 7 terminal commands such as cd, help, exit, mkdir, rmdir, cp, mv. We are proud of everything we have accomplished in this project as it took immense effort and time to put everything together.

5.2 Relevance

In an operating systems course, I think shell scripting is a very relevant piece of work as a file browsers is a major feature of any OS.

5.3 Future Work

- 1) We hope to be able to provide backward compatibility. So when you go back on Nautilus, the terminal automatically calls cd..
- 2) Work on one Nautilus and not have it reload every time we change a directory
- 3) Upon doing the usability study I recognize that experienced users of terminal too appreciate the added feedback. They just did not want Nautilus taking up room as they were aware of their browsing presence. To fix this I thought I could have Nautilus popping up as an option when one runs the program. I figure that would satisfy both sides of the spectrum.

Contributions of team members:

Shashank Gupta- 1) Mkdir and rmdir implementation.

- 2) Set Nautilus up with the program.
- 3) Implementing the icon.
- 4) Edited the final draft of the project report

Rong Xie- 1) cp and mv implementations.

- 2) Fixed a lot of bugs.
- 3) Project report draft.

We together conducted the usability study.

References:

[1] Brennan, S. (n.d.). Tutorial - Write a Shell in C - Stephen Brennan. Retrieved April 07, 2017, from <https://brennan.io/2015/01/16/write-a-shell-in-c/>

[2] Desktop files: putting your application in the desktop menus. (n.d.). Retrieved April 07, 2017, from <https://developer.gnome.org/integration-guide/stable/desktop-files.html.en>

[3] Ubuntu manuals. (n.d.). Retrieved April 07, 2017, from <http://manpages.ubuntu.com/manpages/trusty/man1/gnome-open.1.html>

[4] Köhler, U. (n.d.). How to use mkdir() from sys/stat.h. Retrieved April 07, 2017, from <https://techoverflow.net/2013/04/05/how-to-use-mkdir-from-sysstat-h/>