

```
1 // C Program to design a shell in Linux
2 #include<stdio.h>
3 #include<string.h>
4 #include<stdlib.h>
5 #include<unistd.h>
6 #include<sys/types.h>
7 #include<sys/wait.h>
8 #include<readline/readline.h>
9 #include<readline/history.h>
10
11 #define MAXCOM 1000 // max number of letters to be supported
12 #define MAXLIST 100 // max number of commands to be supported
13
14 // Clearing the shell using escape sequences
15 #define clear() printf("\033[H\033[J")
16
17 // Greeting shell during startup
18 void init_shell()
19 {
20     clear();
21     printf("\n\n\n\n");
22     char* username = getenv("USER");
23     printf("\n\n\nUSER is: %s", username);
24     printf("\n");
25     sleep(1);
26     clear();
27 }
28
29 // Function to take input
30 int takeInput(char* str)
31 {
32     char* buf;
33
34     buf = readline("\n>>> ");
35     if (strlen(buf) != 0) {
36         add_history(buf);
37         strcpy(str, buf);
38         return 0;
39     } else {
40         return 1;
41     }
42 }
43
44 // Function to print Current Directory.
45 void printDir()
46 {
47     char cwd[1024];
48     getcwd(cwd, sizeof(cwd));
49     printf("\nDir: %s", cwd);
50 }
51
52 // Function where the system command is executed
53 void execArgs(char** parsed)
54 {
55     // Forking a child
56     pid_t pid = fork();
57
58     if (pid == -1) {
59         printf("\nFailed forking child..");
60         return;
61     } else if (pid == 0) {
62         if (execvp(parsed[0], parsed) < 0) {
63             printf("\nCould not execute command..");
64         }
65         exit(0);
66     } else {
```

```
67         // waiting for child to terminate
68         wait(NULL);
69         return;
70     }
71 }
72
73 // Function where the piped system commands is executed
74 void execArgsPiped(char** parsed, char** parsedpipe)
75 {
76     // 0 is read end, 1 is write end
77     int pipefd[2];
78     pid_t p1, p2;
79
80     if (pipe(pipefd) < 0) {
81         printf("\nPipe could not be initialized");
82         return;
83     }
84     p1 = fork();
85     if (p1 < 0) {
86         printf("\nCould not fork");
87         return;
88     }
89
90     if (p1 == 0) {
91         // Child 1 executing..
92         // It only needs to write at the write end
93         close(pipefd[0]);
94         dup2(pipefd[1], STDOUT_FILENO);
95         close(pipefd[1]);
96
97         if (execvp(parsed[0], parsed) < 0) {
98             printf("\nCould not execute command 1..");
99             exit(0);
100         }
101     } else {
102         // Parent executing
103         p2 = fork();
104
105         if (p2 < 0) {
106             printf("\nCould not fork");
107             return;
108         }
109
110         // Child 2 executing..
111         // It only needs to read at the read end
112         if (p2 == 0) {
113             close(pipefd[1]);
114             dup2(pipefd[0], STDIN_FILENO);
115             close(pipefd[0]);
116             if (execvp(parsedpipe[0], parsedpipe) < 0) {
117                 printf("\nCould not execute command 2..");
118                 exit(0);
119             }
120         } else {
121             // parent executing, waiting for two children
122             wait(NULL);
123             wait(NULL);
124         }
125     }
126 }
127
128 // Help command builtin
129 void openHelp()
130 {
131     puts("\n***WELCOME TO MY SHELL HELP***"
132         "\nCopyright @ Suprotik Dey")
133 }
```

```
133         "\n-Use the shell at your own risk..."
134         "\nList of Commands supported:"
135         "\n>cd"
136         "\n>ls"
137         "\n>exit"
138         "\n>all other general commands available in UNIX shell"
139         "\n>pipe handling"
140         "\n>improper space handling");
141
142     return;
143 }
144
145 // Function to execute builtin commands
146 int ownCmdHandler(char** parsed)
147 {
148     int NoOfOwnCmds = 4, i, switchOwnArg = 0;
149     char* ListOfOwnCmds[NoOfOwnCmds];
150     char* username;
151
152     ListOfOwnCmds[0] = "exit";
153     ListOfOwnCmds[1] = "cd";
154     ListOfOwnCmds[2] = "help";
155     ListOfOwnCmds[3] = "hello";
156
157     for (i = 0; i < NoOfOwnCmds; i++) {
158         if (strcmp(parsed[0], ListOfOwnCmds[i]) == 0) {
159             switchOwnArg = i + 1;
160             break;
161         }
162     }
163
164     switch (switchOwnArg) {
165     case 1:
166         printf("\nGoodbye\n");
167         exit(0);
168     case 2:
169         chdir(parsed[1]);
170         return 1;
171     case 3:
172         openHelp();
173         return 1;
174     case 4:
175         username = getenv("USER");
176         printf("\nHello %s.\nMind that this is "
177             "not a place to play around."
178             "\nUse help to know more..\n",
179             username);
180         return 1;
181     default:
182         break;
183     }
184
185     return 0;
186 }
187
188 // function for finding pipe
189 int parsePipe(char* str, char** strpiped)
190 {
191     int i;
192     for (i = 0; i < 2; i++) {
193         strpiped[i] = strsep(&str, "|");
194         if (strpiped[i] == NULL)
195             break;
196     }
197
198     if (strpiped[1] == NULL)
```

```
199         return 0; // returns zero if no pipe is found.
200     else {
201         return 1;
202     }
203 }
204
205 // function for parsing command words
206 void parseSpace(char* str, char** parsed)
207 {
208     int i;
209
210     for (i = 0; i < MAXLIST; i++) {
211         parsed[i] = strsep(&str, " ");
212
213         if (parsed[i] == NULL)
214             break;
215         if (strlen(parsed[i]) == 0)
216             i--;
217     }
218 }
219
220 int processString(char* str, char** parsed, char** parsedpipe)
221 {
222     char* strpiped[2];
223     int piped = 0;
224
225     piped = parsePipe(str, strpiped);
226
227     if (piped) {
228         parseSpace(strpiped[0], parsed);
229         parseSpace(strpiped[1], parsedpipe);
230     } else {
231         parseSpace(str, parsed);
232     }
233
234     if (ownCmdHandler(parsed))
235         return 0;
236     else
237         return 1 + piped;
238 }
239
240 int main()
241 {
242     char inputString[MAXCOM], *parsedArgs[MAXLIST];
243     char* parsedArgsPiped[MAXLIST];
244     int execFlag = 0;
245     init_shell();
246
247     while (1) {
248         // print shell line
249         printDir();
250         // take input
251         if (takeInput(inputString))
252             continue;
253         // process
254         execFlag = processString(inputString,
255                                 parsedArgs, parsedArgsPiped);
256         // execflag returns zero if there is no command
257         // or it is a builtin command,
258         // 1 if it is a simple command
259         // 2 if it is including a pipe.
260
261         // execute
262     }
```

```
265             if (execFlag == 1)
266                 execArgs(parsedArgs);
267
268             if (execFlag == 2)
269                 execArgsPiped(parsedArgs, parsedArgsPiped);
270         }
271         return 0;
272     }
```