

**INSTITUTE OF TECHNICAL EDUCATION AND
RESEARCH**
(SOA Deemed to be University)

**PROJECT REPORT
ON
Custom Shell Implementation
(LINUX OS)**



Submitted By:

Name: Raj Vardhan Singh

Registration No.: 2241019502

Branch: CSE

Batch: 13

CODE:

```

#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include <optional>
#include <csignal>
#include <cerrno>
#include <cstring>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>

using namespace std;

struct Redir { optional<string> in; optional<string> out; optional<string> err; bool append_out=false; bool append_err=false; };
struct Cmd { vector<string> argv; };
struct Pipeline { vector<Cmd> cmds; Redir r; bool background=false; string raw; vector<pid_t> pids; pid_t pgid=0; };
struct Job { int id; pid_t pgid; string cmd; bool stopped; bool background; vector<pid_t> pids; };

vector<Job> jobs;
int next_job_id=1;
termios shell_tmodes;
pid_t shell_pgid;
volatile sig_atomic_t sigchld_flag=0;

void sigchld_handler(int){ sigchld_flag=1; }

void ignore_job_signals(){ signal(SIGINT, SIG_IGN); signal(SIGTSTP, SIG_IGN);
signal(SIGQUIT, SIG_IGN); signal(SIGTTIN, SIG_IGN); signal(SIGTTOU, SIG_IGN);
signal(SIGCHLD, sigchld_handler); }

void restore_job_signals(){ signal(SIGINT, SIG_DFL); signal(SIGTSTP, SIG_DFL);
signal(SIGQUIT, SIG_DFL); signal(SIGTTIN, SIG_DFL); signal(SIGTTOU, SIG_DFL);
signal(SIGCHLD, SIG_DFL); }

vector<string> tokenize(const string& s){
    vector<string> t; string cur; bool sq=false,dq=false,esc=false;
    for(size_t i=0;i<s.size();++i){
        char c=s[i];

```

```

    if(esc){ cur.push_back(c); esc=false; continue; }
    if(c=='\\') { esc=true; continue; }
    if(c=='\"&&!dq){ sq=!sq; continue; }
    if(c==""&&!sq){ dq=!dq; continue; }
    ifisspace((unsigned char)c) && !sq && !dq){ if(!cur.empty()){ t.push_back(cur);
    cur.clear(); } continue; }
    cur.push_back(c);
}
if(!cur.empty()) t.push_back(cur);
return t;
}

```

```

Pipeline parse_line(const string& line){
    Pipeline p; p.raw=line; auto toks=tokenize(line); vector<vector<string>> segments(1);
    for(size_t i=0;i<toks.size();++i){ if(toks[i]==")") segments.push_back({}); else
    segments.back().push_back(toks[i]); }
    if(!segments.empty() && !segments.back().empty() && segments.back().back()=='&'){
    p.background=true; segments.back().pop_back(); }
    for(auto& seg:segments){
        Cmd c;
        for(size_t i=0;i<seg.size();++i){
            string tk=seg[i];
            if(tk=='<' && i+1<seg.size()){ p.r.in=seg[++i]; continue; }
            if((tk=='>'||tk=='>>') && i+1<seg.size()){ p.r.out=seg[++i];
p.r.append_out=(tk=='>>'); continue; }
            if((tk=='2>'||tk=='2>>') && i+1<seg.size()){ p.r.err=seg[++i];
p.r.append_err=(tk=='2>>'); continue; }
            c.argv.push_back(tk);
        }
        if(!c.argv.empty()) p.cmds.push_back(c);
    }
    return p;
}

```

```

char** build_argv(const vector<string>& v){
    char** a=(char**)calloc(v.size()+1,sizeof(char*));
    for(size_t i=0;i<v.size();++i) a[i]=strdup(v[i].c_str());
    a[v.size()]=nullptr;
    return a;
}

```

```

optional<Job*> find_job_by_id(int id){ for(auto& j:jobs) if(j.id==id) return &j; return
nullopt; }

```

```

void print_jobs(){ for(auto& j:jobs){ string st=j.stopped?"stopped":"running";
cout<<"["<<j.id<<"] "<<j.pgid<<" "<<st<<" \t"<<j.cmd<<"\n"; } }

```

```

void remove_done_jobs(){

```

```

int status;
for(size_t idx=0; idx<jobs.size();){
    bool any_alive=false;
    for(pid_t pid:jobs[idx].pids){
        if(waitpid(pid,&status,WNOHANG|WUNTRACED|WCONTINUED)>0){
            if(WIFSTOPPED(status)) jobs[idx].stopped=true;
            if(WIFCONTINUED(status)) jobs[idx].stopped=false;
        }
        if(kill(pid,0)==0) any_alive=true;
    }
    if(!any_alive){
        idx=jobs.erase(jobs.begin()+idx)-jobs.begin();
    }else{
        idx++;
    }
}
}

void reap_children(){
    if(!sigchld_flag) return;
    sigchld_flag=0;
    int status;
    while(true){
        pid_t pid=waitpid(-1,&status,WNOHANG|WUNTRACED|WCONTINUED);
        if(pid<=0) break;
        for(auto& j:jobs){
            if(find(j.pids.begin(),j.pids.end(),pid)!=j.pids.end()){
                if(WIFSTOPPED(status)) j.stopped=true;
                if(WIFCONTINUED(status)) j.stopped=false;
            }
        }
        remove_done_jobs();
    }
}

int open_redir(const string& path,int flags, bool append){
    int f=flags;
    if(((flags      &      O_WRONLY)      ||      (flags      &      O_RDWR))) f|=O_CREAT|(append?O_APPEND:O_TRUNC);
    int fd=open(path.c_str(),f,0644);
    return fd;
}

int exec_pipeline(Pipeline& pl){
    size_t n=pl.cmds.size(); if(n==0) return 0;
    vector<int> pipes(max((size_t)0,n-1)*2,-1);
    for(size_t i=0;i+1<n;++i){
        int fds[2];

```

```

if(pipe(fds)==-1){ perror("pipe"); return -1; }
pipes[2*i]=fds[0];
pipes[2*i+1]=fds[1];
}
pid_t pgid=0; pl.pids.clear();
for(size_t i=0;i<n;++i){
    pid_t pid=fork();
    if(pid<0){ perror("fork"); return -1; }
    if(pid==0){
        restore_job_signals();
        if(pgid==0) setpgid(0,0); else setpgid(0,pgid);
        if(i>0) dup2(pipes[2*(i-1)], STDIN_FILENO);
        if(i+1<n) dup2(pipes[2*i+1], STDOUT_FILENO);
        for(size_t k=0;k<pipes.size();++k){ if(pipes[k]!=-1) close(pipes[k]); }
        if(pl.r.in && i==0){
            int fd=open_redir(*pl.r.in,O_RDONLY,false);
            if(fd<0){ perror("open"); _exit(1); }
            dup2(fd,STDIN_FILENO); close(fd);
        }
        if(pl.r.out && i==n-1){
            int fd=open_redir(*pl.r.out,O_WRONLY,pl.r.append_out);
            if(fd<0){ perror("open"); _exit(1); }
            dup2(fd,STDOUT_FILENO); close(fd);
        }
        if(pl.r.err && i==n-1){
            int fd=open_redir(*pl.r.err,O_WRONLY,pl.r.append_err);
            if(fd<0){ perror("open"); _exit(1); }
            dup2(fd,STDERR_FILENO); close(fd);
        }
        char** a=build_argv(pl.cmds[i].argv);
        execvp(a[0],a);
        perror("execvp");
        _exit(127);
    }else{
        if(pgid==0) pgid=pid;
        setpgid(pid,pgid);
        pl.pids.push_back(pid);
    }
}
for(size_t k=0;k<pipes.size();++k){
    if(pipes[k]!=-1) close(pipes[k]);
}
pl.pgid=pgid;
Job j{next_job_id++, pgid, pl.raw, false, pl.background, pl.pids};
jobs.push_back(j);
if(!pl.background){
    tcsetpgrp(STDIN_FILENO, pgid);
    int status; bool stopped=false;
}

```

```

for(pid_t cpid:pl.pids){
    while(true){
        pid_t w=waitpid(cpid,&status,WUNTRACED);
        if(w== -1){
            if(errno==EINTR) continue;
            break;
        }
        if(WIFSTOPPED(status)){ stopped=true; break; }
        if(WIFEXITED(status)||WIFSIGNALED(status)) break;
    }
}

tcsetpgrp(STDIN_FILENO, shell_pgid);
tcsetattr(STDIN_FILENO, TCSADRAIN, &shell_tmodes);
if(!stopped){
    jobs.erase(remove_if(jobs.begin(),jobs.end(),[&](const Job& x){return x.pgid==pgid;}),jobs.end());
}
}else{
    cout<<"["<<jobs.back().id<<"] "<<pgid<<"\n";
}
return 0;
}

bool is_number(const string& s){ if(s.empty()) return false; return all_of(s.begin(),s.end(),::isdigit); }

int builtin_cd(const vector<string>& args){
    const char* path = args.size()>1? args[1].c_str() : getenv("HOME");
    if(!path) path="/";
    if(chdir(path)!=0){ perror("cd"); return 1; }
    return 0;
}

int builtin_pwd(){ char buf[4096]; if(getcwd(buf,sizeof(buf))) cout<<buf<<"\n"; return 0; }

int builtin_jobs(){ reap_children(); print_jobs(); return 0; }

int builtin_fg(const vector<string>& args){
    if(args.size()<2) return 1;
    string a=args[1];
    if(a.size()>0 && a[0]=='%') a=a.substr(1);
    if(!is_number(a)) return 1;
    int id=stoi(a);
    auto oj=find_job_by_id(id); if(!oj) return 1; Job* j=*oj;
    j->background=false;
    tcsetpgrp(STDIN_FILENO, j->pgid);
    kill(-j->pgid, SIGCONT);
    int status;
}

```

```

for(pid_t p:j->pids){
    while(true){
        pid_t w=waitpid(p,&status,WUNTRACED);
        if(w==-1){
            if(errno==EINTR) continue;
            break;
        }
        if(WIFSTOPPED(status)) { j->stopped=true; break; }
        if(WIFEXITED(status)||WIFSIGNALED(status)) break;
    }
}
tcsetpgrp(STDIN_FILENO, shell_pgid);
tcsetattr(STDIN_FILENO, TCSADRAIN, &shell_tmodes);
if(!j->stopped){
    jobs.erase(remove_if(jobs.begin(),jobs.end(),[&](const Job& x){return x.id==j->id;}),jobs.end());
}
return 0;
}

int builtin_bg(const vector<string>& args){
    if(args.size()<2) return 1;
    string a=args[1];
    if(a.size()>0 && a[0]=='%') a=a.substr(1);
    if(!is_number(a)) return 1;
    int id=stoi(a);
    auto oj=find_job_by_id(id); if(!oj) return 1; Job* j=*oj;
    j->background=true; j->stopped=false;
    kill(-j->pgid, SIGCONT);
    cout<<"["<<j->id<<"] "<<j->pgid<<"\n";
    return 0;
}

bool is_builtin(const string& cmd){
    static vector<string> b={"cd","exit","quit","pwd","jobs","fg","bg"};
    return find(b.begin(),b.end(),cmd)!=b.end();
}

int run_builtin(const vector<string>& argv){
    string c=argv[0];
    if(c=="cd") return builtin_cd(argv);
    if(c=="pwd") return builtin_pwd();
    if(c=="jobs") return builtin_jobs();
    if(c=="fg") return builtin_fg(argv);
    if(c=="bg") return builtin_bg(argv);
    if(c=="exit"||c=="quit"){ cout<<"bye\n"; exit(0); }
    return 0;
}

```

```

void setup_shell(){
    shell_pgid=getpid();
    setpgid(0,0);
    tcgetattr(STDIN_FILENO,&shell_tmodes);
    tcsetpgrp(STDIN_FILENO, shell_pgid);
}

int main(){
    if(isatty(STDIN_FILENO)) setup_shell();
    ignore_job_signals();
    string line;
    while(true){
        reap_children();
        char cwd[4096];
        if(!getcwd(cwd,sizeof(cwd))){
            perror("getcwd");
            cwd[0]='\0';
        }
        cout<<"mini:"<<cwd<<"$ "<<flush;
        if(!getline(cin,line)){
            cout<<"\n";
            break;
        }
        if(line.size()==0) continue;
        Pipeline pl=parse_line(line);
        if(pl.cmds.empty()) continue;
        if(is_builtin(pl.cmds[0].argv[0]) && pl.cmds.size()==1 && !pl.r.in && !pl.r.out && !pl.r.err){
            run_builtin(pl.cmds[0].argv);
            continue;
        }
        exec_pipeline(pl);
    }
    return 0;
}

```

Screenshots

Code:

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 #include <sstream>
5 #include <algorithm>
6 #include <optional>
7 #include <csignal>
8 #include <cerrno>
9 #include <cstring>
10 #include <unistd.h>
11 #include <sys/types.h>
12 #include <sys/wait.h>
13 #include <sys/stat.h>
14 #include <fcntl.h>
15 #include <termios.h>
16
17 using namespace std;
18
19 struct Redir { optional<string> in; optional<string>
    out; optional<string> err; bool append_out=false; bool
    append_err=false; };
20 struct Cmd { vector<string> argv; };
21 struct Pipeline { vector<Cmd> cmds; Redir r; bool
    background=false; string raw; vector<pid_t> pids; pid_t
    pgid=0; };
22 struct Job { int id; pid_t pgid; string cmd; bool
    stopped; bool background; vector<pid_t> pids; };
23
24 vector<Job> jobs;
25 int next_job_id=1;
26 termios shell_tmodes;
27 pid_t shell_pgid;
28 volatile sig_atomic_t sigchld_flag=0;
29
30 void sigchld_handler(int){ sigchld_flag=1; }
31
32 void ignore_job_signals(){ signal(SIGINT, SIG_IGN); signal
    (SIGTSTP, SIG_IGN); signal(SIGQUIT, SIG_IGN); signal
    (SIGTTIN, SIG_IGN); signal(SIGTTOU, SIG_IGN); signal
    (SIGCHLD, sigchld_handler); }
```

```
1
2 void restore_job_signals(){ signal(SIGINT, SIG_DFL);
3   signal(SIGTSTP, SIG_DFL); signal(SIGQUIT, SIG_DFL); signal
4   (SIGTTIN, SIG_DFL); signal(SIGTTOU, SIG_DFL); signal
5   (SIGCHLD, SIG_DFL); }
6
7 vector<string> tokenize(const string& s){
8     vector<string> t; string cur; bool sq=false,dq=false
9     ,esc=false;
10    for(size_t i=0;i<s.size();++i){
11        char c=s[i];
12        if(esc){ cur.push_back(c); esc=false; continue; }
13        if(c=='\\'){ esc=true; continue; }
14        if(c=='\'&&!dq){ sq=!sq; continue; }
15        if(c=='\"&&!sq){ dq=!dq; continue; }
16        if(ispace((unsigned char)c) && !sq && !dq){ if(!
17 cur.empty()){ t.push_back(cur); cur.clear(); } continue; }
18        cur.push_back(c);
19    }
20    if(!cur.empty()) t.push_back(cur);
21    return t;
22 }
23
24 Pipeline parse_line(const string& line){
25     Pipeline p; p.raw=line; auto toks=tokenize
26     (line); vector<vector<string>> segments(1);
27     for(size_t i=0;i<toks.size();++i){ if(toks[i]=="|")
28         segments.push_back({}); else segments.back().push_back(
29         toks[i]); }
30     if(!segments.empty() && !segments.back().empty() &&
31     segments.back().back()=="&"){ p.background=true; segments.
32     back().pop_back(); }
33     for(auto& seg:segments){
34         Cmd c;
35         for(size_t i=0;i<seg.size();++i){
36             string tk=seg[i];
37             if(tk=="<" && i+1<seg.size()){ p.r.in=seg[++
38                 i]; continue; }
```

```
1 if((tk==>"||tk==>>") && i+1<seg.size()){ p.r.out=seg[++  
i]; p.r.append_out=(tk==>>); continue; }  
2 if((tk=="2>"||tk=="2>>") && i+1<seg.size()){ p  
.r.err=seg[++i]; p.r.append_err=(tk=="2>>"); continue; }  
3 c.argv.push_back(tk);  
4 }  
5 if(!c.argv.empty()) p.cmds.push_back(c);  
6 }  
7 return p;  
8 }  
9  
10 char** build_argv(const vector<string>& v){  
11     char** a=(char**)calloc(v.size()+1,sizeof(char*));  
12     for(size_t i=0;i<v.size();++i) a[i]= strdup(v[i].c_str  
());  
13     a[v.size()]=nullptr;  
14     return a;  
15 }  
16  
17 optional<Job*> find_job_by_id(int id){ for(auto& j:jobs)  
if(j.id==id) return &j; return nullopt; }  
18  
19 void print_jobs(){ for(auto& j:jobs){ string st=j.stopped?  
"stopped":"running"; cout<< "[" <<j.id<<"] " <<j.pgid<<" "<<  
st<<" \t"<<j.cmd<<"\n"; } }  
20  
21 void remove_done_jobs(){  
22     int status;  
23     for(size_t idx=0; idx<jobs.size();){  
24         bool any_alive=false;  
25         for(pid_t pid:jobs[idx].pids){  
26             if(waitpid(pid,&status,WNOHANG|WUNTRACED|  
WCONTINUED)>0){  
27                 if(WIFSTOPPED(status)) jobs[idx].stopped=  
true;  
28                 if(WIFCONTINUED(status)) jobs[idx].stopped  
=false;  
29             }  
        }  
    }  
}
```



```
1             if(kill(pid,0)==0) any_alive=true;
2         }
3         if(!any_alive){
4             idx=jobs.erase(jobs.begin()+idx)-jobs.begin();
5         }else{
6             idx++;
7         }
8     }
9 }
10
11 void reap_children(){
12     if(!sigchld_flag) return;
13     sigchld_flag=0;
14     int status;
15     while(true){
16         pid_t pid=waitpid(-1,&status,WNOHANG|WUNTRACED|
17         WCONTINUED);
18         if(pid<=0) break;
19         for(auto& j:jobs){
20             if(find(j.pids.begin(),j.pids.end(),pid)!=j.
21             pids.end()){
22                 if(WIFSTOPPED(status)) j.stopped=true;
23                 if(WIFCONTINUED(status)) j.stopped=false;
24             }
25         }
26         remove_done_jobs();
27     }
28     int open_redir(const string& path,int flags, bool append){
29         int f=flags;
30         if(((flags & O_WRONLY) || (flags & O_RDWR))) f|=
31             O_CREAT|(append?O_APPEND:O_TRUNC);
32         int fd=open(path.c_str(),f,0644);
33         return fd;
34 }
```



```
1 int exec_pipeline(Pipeline& pl){
2     size_t n=pl.cmds.size(); if(n==0) return 0;
3     vector<int> pipes(max((size_t)0,n-1)*2,-1);
4     for(size_t i=0;i+1<n;++i){
5         int fds[2];
6         if(pipe(fds)==-1){ perror("pipe"); return -1; }
7         pipes[2*i]=fds[0];
8         pipes[2*i+1]=fds[1];
9     }
10    pid_t pgid=0; pl.pids.clear();
11    for(size_t i=0;i<n;++i){
12        pid_t pid=fork();
13        if(pid<0){ perror("fork"); return -1; }
14        if(pid==0){
15            restore_job_signals();
16            if(pgid==0) setpgid(0,0); else setpgid(0
17 ,pgid);
18            if(i>0) dup2(pipes[2*(i-1)], STDIN_FILENO);
19            if(i+1<n) dup2(pipes[2*i+1], STDOUT_FILENO);
20            for(size_t k=0;k<pipes.size();++k){ if(pipes
21 [k]!=-1) close(pipes[k]); }
22            if(pl.r.in && i==0){
23                int fd=open_redir(*pl.r.in,O_RDONLY,false
24 );
25                if(fd<0){ perror("open"); _exit(1); }
26                dup2(fd,STDIN_FILENO); close(fd);
27            }
28            if(pl.r.out && i==n-1){
29                int fd=open_redir(*pl.r.out,O_WRONLY,pl.r.
30 append_out);
31                if(fd<0){ perror("open"); _exit(1); }
32                dup2(fd,STDOUT_FILENO); close(fd);
33            }
34            if(pl.r.err && i==n-1){
35                int fd=open_redir(*pl.r.err,O_WRONLY,pl.r.
36 append_err);
37                if(fd<0){ perror("open"); _exit(1); }
```



```
1          dup2(fd, STDERR_FILENO); close(fd);
2      }
3      char** a=build_argv(pl.cmds[i].argv);
4      execvp(a[0],a);
5      perror("execvp");
6      _exit(127);
7 }else{
8     if(pgid==0) pgid=pid;
9     setpgid(pid,pgid);
10    pl.pids.push_back(pid);
11 }
12 }
13 for(size_t k=0;k
```



```
1     if(!stopped){
2         jobs.erase(remove_if(jobs.begin(),jobs.end(),
3             &](const Job& x){return x.pgid==pgid;}),jobs.end());
4     }
5     else{
6         cout<<"["<<jobs.back().id<<"] "<<pgid<<"\n";
7     }
8     return 0;
9
10 bool is_number(const string& s){ if(s.empty()) return
11     false; return all_of(s.begin(),s.end(),::isdigit); }
12
13 int builtin_cd(const vector<string>& args){
14     const char* path = args.size()>1? args[1].c_str() :
15         getenv("HOME");
16     if(!path) path="/";
17     if(chdir(path)!=0){ perror("cd"); return 1; }
18     return 0;
19 }
20
21 int builtin_pwd(){ char buf[4096]; if(getcwd(buf,sizeof
22     (buf))) cout<<buf<<"\n"; return 0; }
23
24 int builtin_fg(const vector<string>& args){
25     if(args.size()<2) return 1;
26     string a=args[1];
27     if(a.size()>0 && a[0]=='%') a=a.substr(1);
28     if(!is_number(a)) return 1;
29     int id=stoi(a);
30     auto obj=find_job_by_id(id); if(!obj) return 1; Job* j=*
31     obj;
32     j->background=false;
33     tcsetpgrp(STDIN_FILENO, j->pgid);
```



```
1     kill(-j->pgid, SIGCONT);
2     int status;
3     for(pid_t p:j->pids){
4         while(true){
5             pid_t w=waitpid(p,&status,WUNTRACED);
6             if(w==-1){
7                 if(errno==EINTR) continue;
8                 break;
9             }
10            if(WIFSTOPPED(status)) { j->stopped=true;
11                break; }
12            if(WIFEXITED(status)||WIFSIGNALED(status))
13                break;
14        }
15        tcsetpgrp(STDIN_FILENO, shell_pgid);
16        tcsetattr(STDIN_FILENO, TCSADRAIN, &shell_tmodes);
17        if(!j->stopped){
18            jobs.erase(remove_if(jobs.begin(),jobs.end(),[&](
19                const Job& x){return x.id==j->id;}),jobs.end());
20        }
21    }
22    int builtin_bg(const vector<string>& args){
23        if(args.size()<2) return 1;
24        string a=args[1];
25        if(a.size()>0 && a[0]=='%') a=a.substr(1);
26        if(!is_number(a)) return 1;
27        int id=stoi(a);
28        auto obj=find_job_by_id(id); if(!obj) return 1; Job* j=*
29        obj;
30        j->background=true; j->stopped=false;
31        kill(-j->pgid, SIGCONT);
32        cout<<"["<<j->id<<"] "<<j->pgid<<"\n";
33    }
```



```
1 bool is_builtin(const string& cmd){ static vector<string>
2     b={"cd","exit","quit","pwd","jobs","fg","bg"}; return
3     find(b.begin(),b.end(),cmd)!=b.end(); }
4
5 int run_builtin(const vector<string>& argv){
6     string c=argv[0];
7     if(c=="cd") return builtin_cd(argv);
8     if(c=="pwd") return builtin_pwd();
9     if(c=="jobs") return builtin_jobs();
10    if(c=="fg") return builtin_fg(argv);
11    if(c=="bg") return builtin_bg(argv);
12    if(c=="exit"||c=="quit"){ cout<<"bye\n"; exit(0); }
13    return 0;
14 }
15
16 void setup_shell(){
17     shell_pgid=getpid();
18     setpgid(0,0);
19     tcgetattr(STDIN_FILENO,&shell_tmodes);
20     tcsetpgrp(STDIN_FILENO, shell_pgid);
21 }
22
23 int main(){
24     if(isatty(STDIN_FILENO)) setup_shell();
25     ignore_job_signals();
26     string line;
27     while(true){
28         reap_children();
29         char cwd[4096];
30         if(!getcwd(cwd,sizeof(cwd))){
31             perror("getcwd");
32             cwd[0]='\0';
33         }
34         cout<<"mini:"<<cwd<<"$ "<<flush;
35         if(!getline(cin,line)){
36             cout<<"\n";
37             break;
38     }
39 }
```

```
1      }
2      if(line.size()==0) continue;
3      Pipeline pl=parse_line(line);
4      if(pl.cmds.empty()) continue;
5      if(is_builtin(pl.cmds[0].argv[0]) && pl.cmds.size
6          ()==1 && !pl.r.in && !pl.r.out && !pl.r.err){
7          run_builtin(pl.cmds[0].argv);
8          continue;
9      }
10     exec_pipeline(pl);
11 }
12 }
13 }
```

Output:

```
raj180@Ubuntu:~/Desktop/Wipro$ ^C
raj180@Ubuntu:~/Desktop/Wipro$ g++ -std=c++17 -O2 -Wall Assignment2.cpp -o Assignment2
raj180@Ubuntu:~/Desktop/Wipro$ ./Assignment2
mini:/home/raj180/Desktop/Wipro$ sleep 30

^Zmini:/home/raj180/Desktop/Wipro$ jobs
[1] 50933 running      sleep 30
mini:/home/raj180/Desktop/Wipro$ sudo whoami
[sudo] password for raj180:
root
mini:/home/raj180/Desktop/Wipro$ ls
Assignment1      Assignment2      Assignment3      backup.sh    suite.sh
Assignment1.cpp  Assignment2.cpp  Assignment3.cpp  logwatch.sh update_cleanup.sh
mini:/home/raj180/Desktop/Wipro$ cd ..
cd: No such file or directory
mini:/home/raj180/Desktop/Wipro$ cd ..
mini:/home/raj180/Desktop$ ls
Project Wipro
mini:/home/raj180/Desktop$ cd Wipro
mini:/home/raj180/Desktop/Wipro$ ls
Assignment1      Assignment2      Assignment3      backup.sh    suite.sh
Assignment1.cpp  Assignment2.cpp  Assignment3.cpp  logwatch.sh update_cleanup.sh
mini:/home/raj180/Desktop/Wipro$ exit
bye
raj180@Ubuntu:~/Desktop/Wipro$
```