The main loop:

```
while(1) {

struct command_t *command = malloc (sizeof(struct command_t));
memset(command, 0, sizeof(struct command_t));          //command created

code = prompt(command);      //handles user input with show_prompt()
                               and parse_command()

code = process_command(command);      //command executed

free_command(command);      //free memory

}
```

- prompt() handles messy keyboard input
- parse_command() takes user input like “ls -la | grep foo > output.txt” and breaks it into structured **command_t** structs.

```
struct command_t {

        char *name; //e.g. “ls”
        bool background;
        bool auto_complete;
        int arg_count;
        char **args;              //e.g. [“ls”, “-la”, NULL]
        char *redirects[3];       // [0] = input(<) [1]= output(>) [2]= append(>>)
        struct command_t *next;   //points to next command if piped
```

if user types ls -la :

- command->name = “ls”
- command->args = [“ls”,”-la”, NULL]
- command->background = false
- command->redirects = all NULL
- command->next = NULL

process_command() :

```
int process_command( struct command_t *command) {

        if (strcmp( command->name , "") == 0) return SUCCESS;          //empty input

        if (strcmp( command->name , "exit") == 0) return EXIT;

        if(strcmp( command->name, "cd") == 0) {
          chdir( command-> args[1]);
          return SUCCESS;
        }


        pid_t pid = fork();

        if (pid == 0) {       // child process

             execvp( command->name , command->args);
             printf("command not found\n");
             exit(127);

        }
        else {                // parent process
            wait(0);
            return SUCCESS;
        }
}
```

Notes:

- After fork(), Parent process (shell) -> child process (copy) : becomes something like "ls" "the command that was typed"
- execv() vs execvp() : both replace the child process with new program but:
    - execvp("ls", args) -> automatically finds "ls" by searching path
    - execv("/bin/ls", args) -> full path must be given as input

- echo $PATH returns a list like /usr/local/sbin:/usr/local/bin:/usr/bin:/sbin:/bin
    - **:** is the separator, its a list because programs are installed in different places and by having a list Linux checks each folder one by one until it finds the program we asked for.
    - when "gcc" is typed :
        - Linux looks in /usr/local/sbin no file continue;

- Linux looks in /usr/local/bin if there is a file called gcc -> run it.
- Even if there are more than one potential matches across the list of folders Linux uses the first one it finds so order of folders in PATH matters!