The cluster shell application is designed in such a way that every client has a corresponding server. The mapping of the name to IP addresses of the server is stored in the cluster_config.txt file. The file does not store information about the port numbers, as the server is assumed to run on a fixed port number – 8000. The communication between the client and any server happens over the network via the TCP protocol.

The client simply runs an infinite loop to keep accepting commands from the user. One can exit the client by using the SIGINT or SIGQUIT signals. It loads the mapping between the name and IP addresses from the config file into a local structure at startup. Every command entered in the terminal is first checked for the presence of pipes. If the pipe operator is present, commands are extracted from left to right. Each individual command is checked for the presence of the name of the machine on which it must be executed. If no name is found, the command is sent to the local server for execution via a socket. Otherwise, the local structure is referred to get the IP address of the requested machine, and the command is sent to the respective server. The socket is shutdown after all the data has been sent. If there was no pipe operator present, the output received from the server is directly displayed to the user. Otherwise, the result is stored in a temporary variable, and then given along with the next command to the corresponding server. We have not used pipes for I/O redirection, as all processing takes place in a single process, and hence the use of pipes was unnecessary. Creation of multiple processes was also avoided as the actual command execution happens on the server side, and due to the stop-and-wait model of a shell program, background processing was not desired.

If a command is to be executed on all the servers (name is of the form n*), the client concurrently sends the command to all the servers and waits till either a response has been received from each server, or the connect call to the server has failed. Similarly, when the user enters the '*nodes'* command, the client tries to connect to all the servers concurrently. If the connect system call is successful, it prints the details of the server, otherwise it assumes that the server is offline.

The server has a passive TCP socket setup at port 8000, to listen for incoming connections. It also stores a config file named server_config.txt to store which client is currently in which folder of the server. When a connection request is initiated, it accepts the connection request, and then checks the command to be executed. If the command is not a cd command, a new process is created to handle the command. The child process reads the input from the socket, if any, and then creates another child process to execute the command. Two separate pipes are used to transfer input and output between the two processes. The process then writes the output to the socket and closes the connection. The cd command is not executed in a child process to avoid race conditions while updating the server_config.txt file.

Structure of messages:

- Client to Server:

Command Length	Command	Input (Optional)

- Server to Client:

Result Length	Result

Execution of commands (Other than cd):