DISTRIBUTED COMPUTING  
**DISTRIBUTED MOBILE STORAGE**

horizontal line

# 

# how-does-remotefast-for-windows-server-solve-the-distributed-file-storage-challenge.jpg

Shashi Shekhar

Raja Nageswara Rao Gogineni

# Introduction

We aimed at creating in-memory distributed file storage using the concepts of consistent hashing and chord. Although initial proposal consists of only consistent hashing. But since we had time we also implemented peer to peer mechanism using chord system. However it turns out consistent hashing is more efficient than chord in our implementations. So we switched back to consistent hashing. As we go through the project I will try to explain the differences in the implementations of chord and consistent hashing although each system has its own advantages.

## Concepts used

1. Distributed hash table
2. Consistent hashing
3. Chord(Peer to peer)

1. Distributed hash table

“A distributed hash table (DHT) is a class of a decentralized distributed system that provides a lookup service similar to ahash table : (*key*, *value*) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows a DHT toscale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.”(Definition as per wiki)

Basically, a distributed hash table behaves like a hash table. Storage is in the form of key value pair and keys are used to retrieve values. Values need not be necessarily stored on disk, however a distributed hash table can be built on top of a persistent hash table. As the name suggests distributed hash tables can be stored and lookups are performed on multiple machines. Unlike the master and slave database replication architectures,in this implementation all nodes are peers that can join and leave the network freely. Because peers can be added and removed anytime in this implementation there will some problems,but distributed hash tables make provable guarantees about performance.

2. Consistent hashing

Consistent hashing is based on mapping each object to a point on the edge of a circle (or equivalently, mapping each object to a real angle). The system maps each available machine (or other storage bucket) to many pseudo-randomly distributed points on the edge of the same circle.

To find where an object should be placed, the system finds the location of that object's key on the edge of the circle; then walks around the circle until falling into the first bucket it encounters (or equivalently, the first available bucket with a higher angle). The result is that each bucket contains all the resources located between its point and the previous bucket point.

If a bucket becomes unavailable (for example because the computer it resides on is not reachable), then the angles it maps to will be removed. Requests for resources that would have mapped to each of those points now map to the next highest point. Since each bucket is associated with many pseudo-randomly distributed points, the resources that were held by that bucket will now map to many different buckets. The items that mapped to the lost bucket must be redistributed among the remaining ones, but values mapping to other buckets will still do so and do not need to be moved.

A similar process occurs when a bucket is added. By adding a bucket point, we make any resources between that and the next smaller angle map to the new bucket. These resources will no longer be associated with the previous bucket, and any value previously stored there will not be found by the selection method described above.

The portion of the keys associated with each bucket can be altered by altering the number of angles that bucket maps to.

3. Chord(Peer to peer)

Chord is a protocol and algorithm for a peer-to-peer distributed hash table. As mentioned above distributed hashtable stores key-value pairs by assigning keys to different Nodes. A node will store the values for all the keys for which it is responsible. Chord specifies how keys are assigned to nodes, and how a node can discover the value for a given key by first locating the node responsible for that key.

Using the Chord lookup protocol, nodes and keys are arranged in an identifier circle that has at most 2^m nodes, ranging from 0 to 2^m - 1. (m should be large enough to avoid collision.)

Each node has a *successor* and a *predecessor*. The successor to a node is the next node in the identifier circle in a clockwise direction. The predecessor is counter-clockwise. If there is a node for each possible ID, the successor of node 0 is node 1, and the predecessor of node 0 is node 2^m - 1; however, normally there are "holes" in the sequence. For example, the successor of node 153 may be node 167 (and nodes from 154 to 166 do not exist); in this case, the predecessor of node 167 will be node 153.

The concept of successor can be used for keys as well. The *successor node* of a key k is the first node whose ID equals to k or follows k in the identifier circle, denoted by successor(k). Every key is assigned to (stored at) its successor node, so looking up a key k is to query successor(k).

Since the successor (or predecessor) of a node may disappear from the network (because of failure or departure), each node records a whole segment of the circle adjacent to it, i.e., the r nodes preceding it and the r nodes following it. This list results in a high probability that a node is able to correctly locate its successor or predecessor, even if the network in question suffers from a high failure rate.

**Implementation and Design**

Option given in UI

1. Downloading
2. Uploading
3. Deleting
4. Adding a node server
5. Deleting a node server

How these options are implemented:

1. Downloading
2. Centralized consistent hashing:In this implementation there is a centralized server which contains the list of IP address of the server and the order in which they are connected. This server is only used to lookup IP addresses. So the first step is to identify the correct server in the ring. So the central server contains the total number of node servers connected. Divide the hashed value with total servers to get the index of IP of server from which it has to download. From list get the Ip address and redirect the request to that node server to download the file.
3. Chord:In this implementation centralized server redirects the request to a random peer server. There need not be any centralized server in this implementation however the purpose of this centralized server is explained later. As mentioned earlier each peer server maintains a finger table that has IP addresses and the range of keys the range of keys managed by peer servers whose position are 20, 21, 22, 23 .... 2n nodes away from this server. First each peer will search if the file is present in the server or not. Or else it will check for the peer server which handles the keys that are just less than the file name’s hash value.The this request is redirected to that server and the process is repeated until the server which handles the key of file is found and the file is downloaded.

Different filenames can have same key. So there is conflict. To resolve this conflict the value in key value pair of hashmap will be list of files with same key instead just one file as value.

2. Uploading

The correct server that handles the key for the file names is found just like mentioned above in both Chord and consistent hashing implementation. And file uploaded is stored in the server found and the next server in the list of ip in centralized server in case consistent hashing or first server in the finger tables in case of chord implementation. This way we maintain replicas so as to restore a files even if crash happens. currently we maintain only two replicas for simplicity. File is stored in hashmap so that there is in memory file storage without saving to disk. For storing replicas different hashmap is used for simplicity.

3. Delete

Just like in uploading appropriate server is found and file is deleted from from hashmap and from the hashmap maintaining replicas in the next node server from list or first server in finger table.

4. Add server

When a new server is added all the key are redistributed and load on keys is reduced. and keys in replica table are also redistributed.

Note: from now next server refers to server that can handle key values that are just than current server’s range of keys. In case of consistent hashing this the next server in list of IP in centralized server or first server in its finger table.

5. remove server or crashes

Here if Server is crashed , all the keys are again redistributed and and all the key are redistributed and load on keys is increased . and keys in replica table are also redistributed. Here we are minimizing the number of RMI Calls by finding fixed number of keys

In case of chord if more than one server is lost you need the next server that can handle keys whose value is just more than itself to redistribute keys and this server might not have the handle to the next server in its finger table and you don't know which server handles next least keys. So you have to search all available servers finger tables. This is very inefficient. That is the reason shifted back centralized consistent hashing implementation. So now we discuss implementation of consistent hashing.

Important aspects of our distributed file storage.

1. Availability:

Our chord implementation is highly available as requested can redirected to any peer server. And central server just redirects the request to a random server. This is the purpose of central server in chord implementation. In this case this central server is not a bottleneck. Instead it aids in increasing availability. As at one point several peer servers handling requests. Whereas in our implementation of consistent hashing central server has to redirect all request to correct server. So centralized server can be bottleneck and hence less availability as only central server has to handle all requests.

1. Scalability

Since files are distributed over network and nodes can be added and removed easily there is high scalability in our implementation.

1. Fault tolerance

We maintain a replica number of two. If more than one consecutive servers are lost some files cannot be recovered. But we will recover as many files as possible. In case of consistent hashing fault tolerance can be achieved efficiently than in case of chord.

1. Decentralized peer to peer

In our chord implementation we achieved decentralized peer to peer implementation but we switched back to consistent hashing because of less efficiency to achieve fault tolerance.

1. In memory file transfer similar to spark

Instead of storing file in disk we stored the entire file in the form of its in a hashmap so that there is in-memory distributed storage just like in spark.

Time Complexities

|  |  |  |
| --- | --- | --- |
|  | Consistent hashing | Chord |
| Upload | O(1) | O(log n) |
| Download | O(1) | O(log n) |
| Delete | O(1) | O(log n) |
| Add a server | O(n) | O(log n) |
| Remove server or crash | O(n) | O(n2 log n) |

Major Challenges Faced:

Although we faced a lot of challenges due lack enough sources about chord and during implementation. Only 3 biggest challenges are mentioned.

1. While removing a node or if a crash happened recovery of files is one of the biggest challenges we faced. It is not just recovery of files it also involves redistribution of files and redistribution of files in both replica hashmap and actual hashmap. We implemented fault tolerance of any number of failures. As we had to manage replicas and had to handle any number of failures it was a complex implementation
2. Maintaining replica table while adding or removing a nodes. All the files in replica hashmap and actual hashmap has to be redistributed in one go. To implement this efficiently we faced a lot of problems during file redistribution.
3. We first implemented consistent hashing. Then we tried to implement the chord using the same chord. For this we tried to distribute ip table in centralized server in consistent hashing across finger table in peer server. This was very complex and there were no proper sources. However we shifted back to consistent hashing as we felt it is more efficient in lof aspects than chord.

Future work

1. We already tried to implement in mobile platform however due to lack of enough android mobiles to test and lack of time. We had to postpone this work to future. This could not done on emulators because they do not accept connection external server and buffer cannot be shared between two applications. Hower file transfer using socket server is working fine. So we plan to implement our distributed file storage as a mobile app in future
2. Right now we are maintaining two replicas. However as future work we plan to implement replicas as an input from user, so that file recovery can work for any number of replicas.
3. Instead of just file storage we plan to implement file system which has functionalities like ls, organizing in form of folders, Deleting folders, adding folder, copy and pasting files to different folders etc.

**Snapshot**

Here I am using centralized server and three other node as peer server .

IP of centralized server129.210.16.134

IP of three node are

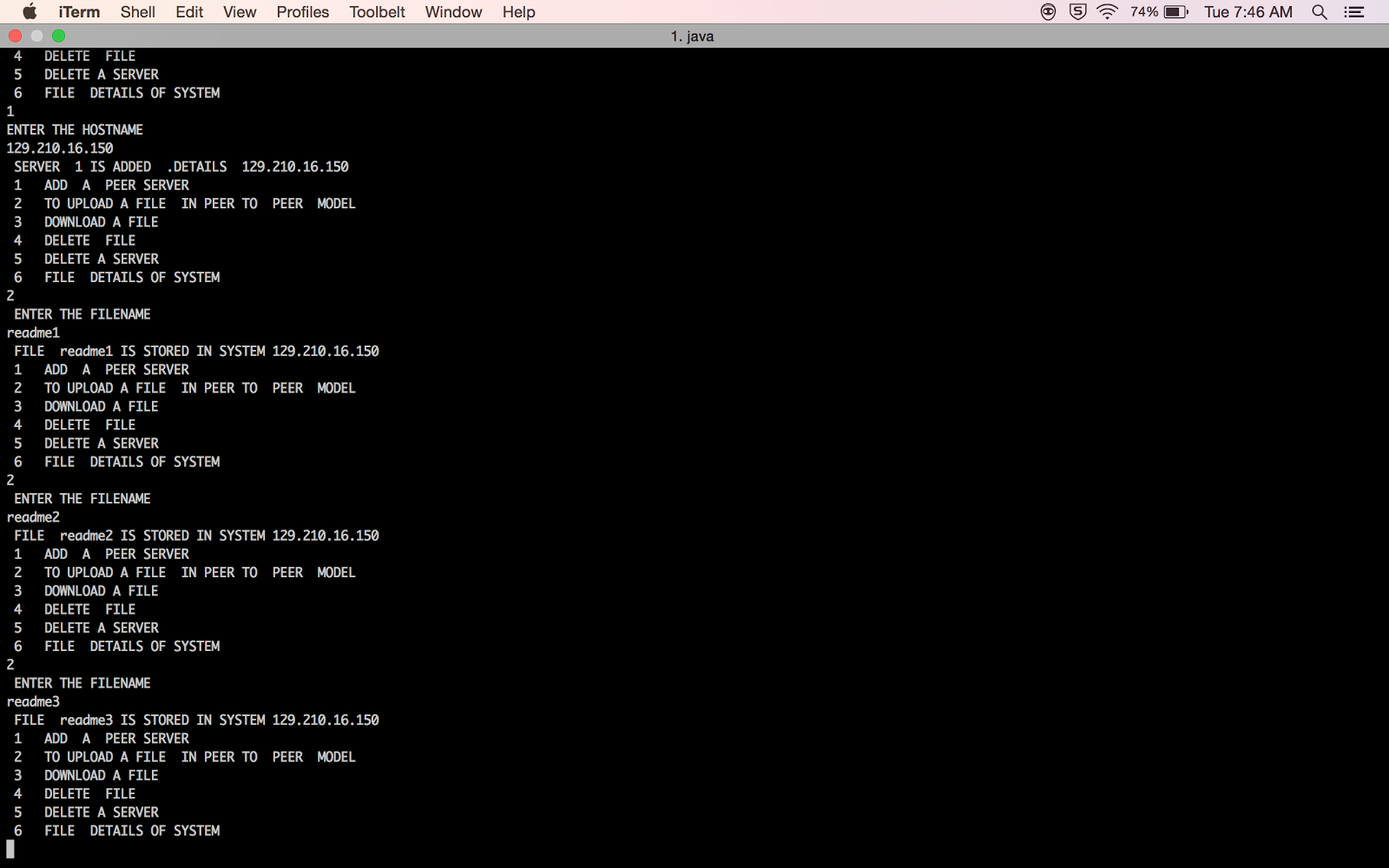
2) 129.210.16.150

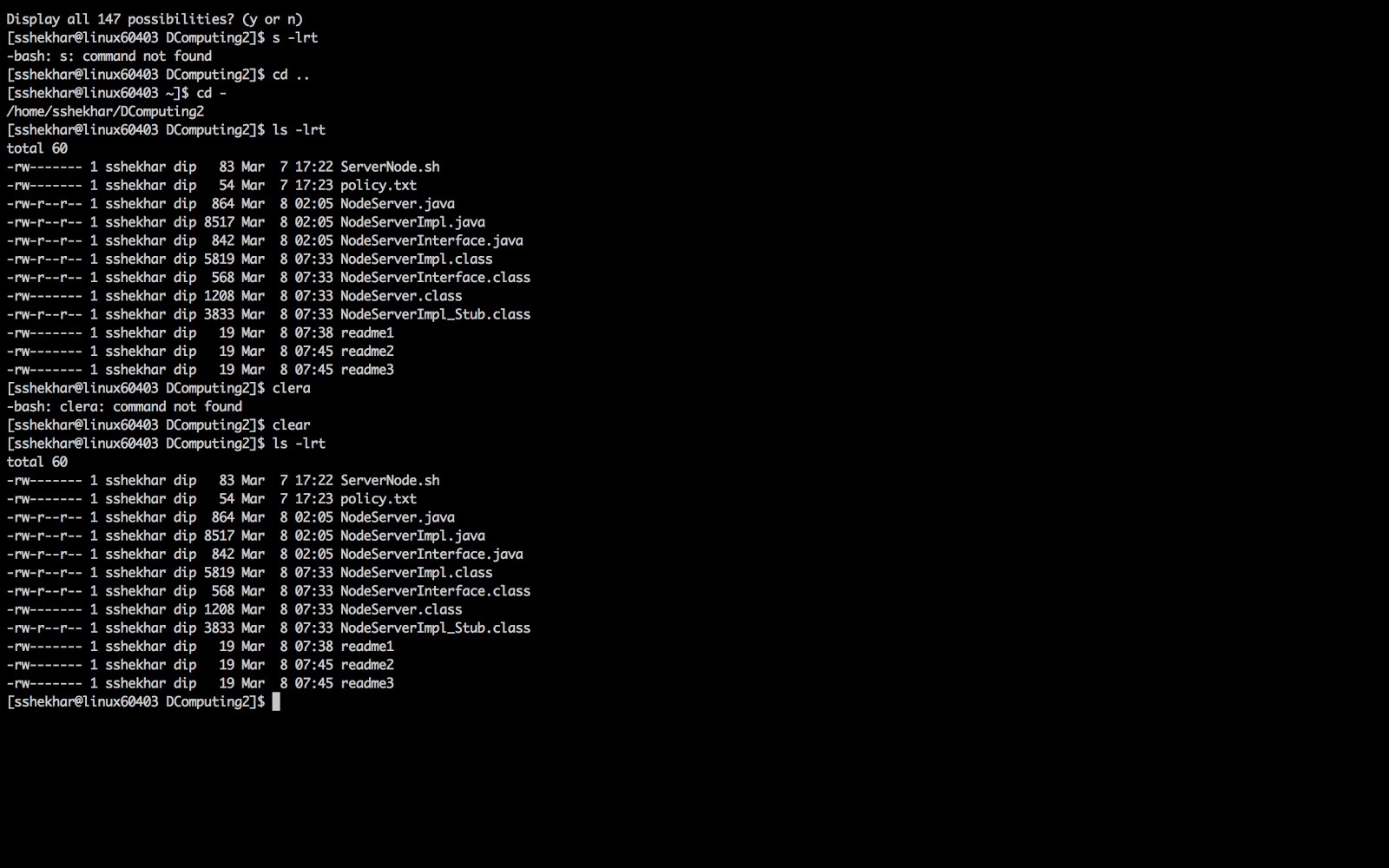
2)129.210.16.134

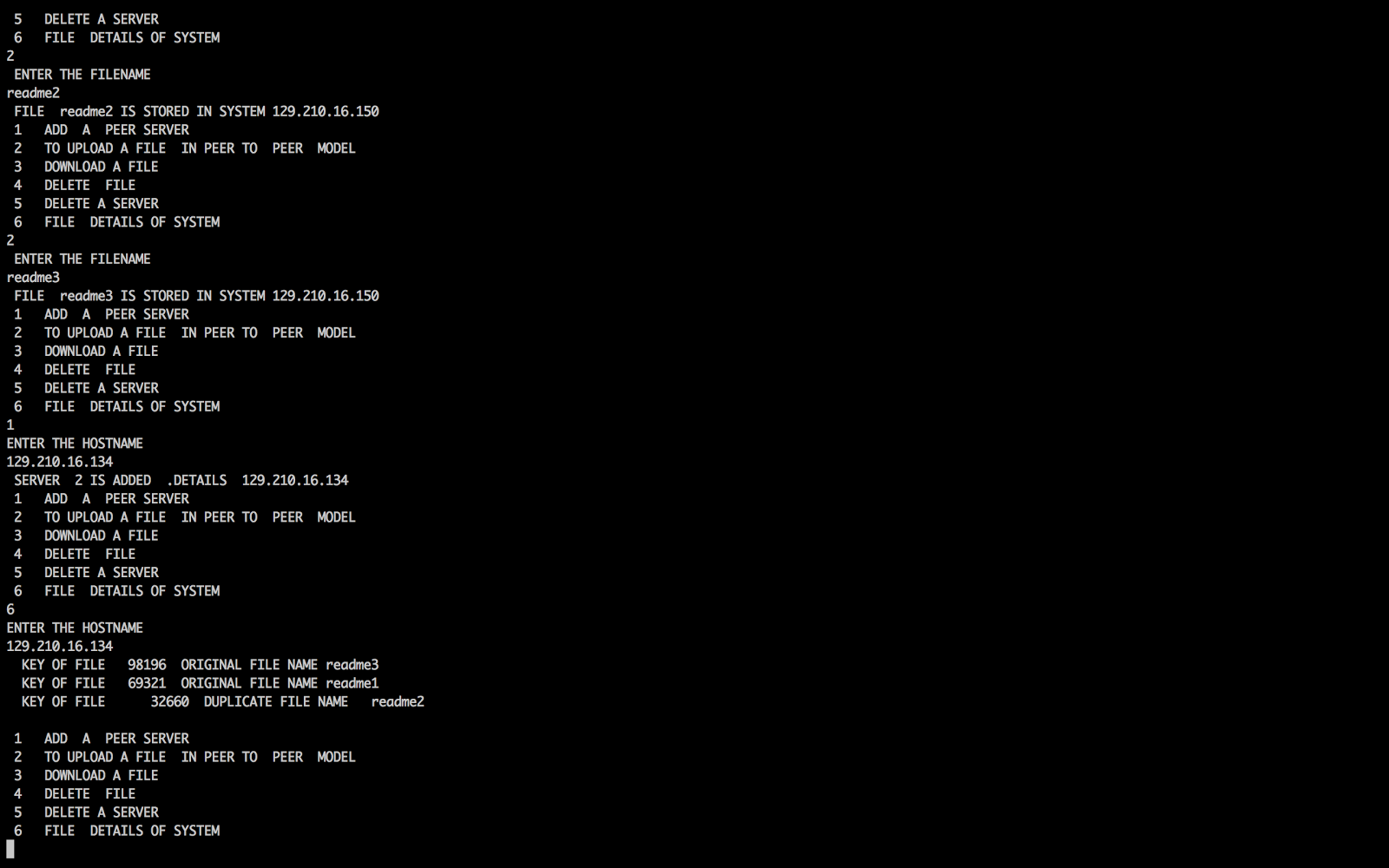
3) 129.210.16.148

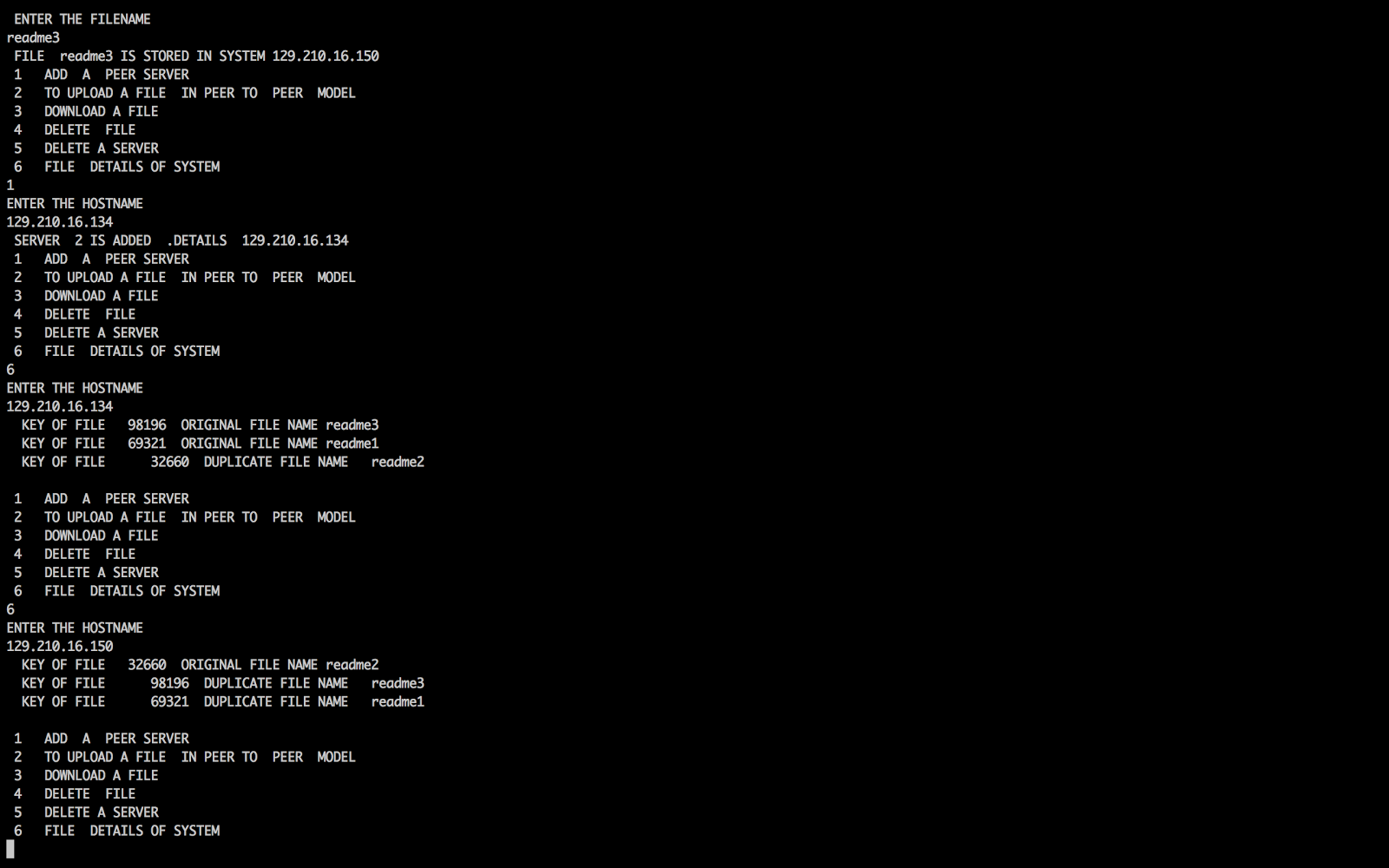
All the nodes must be connected to each other through LAN .

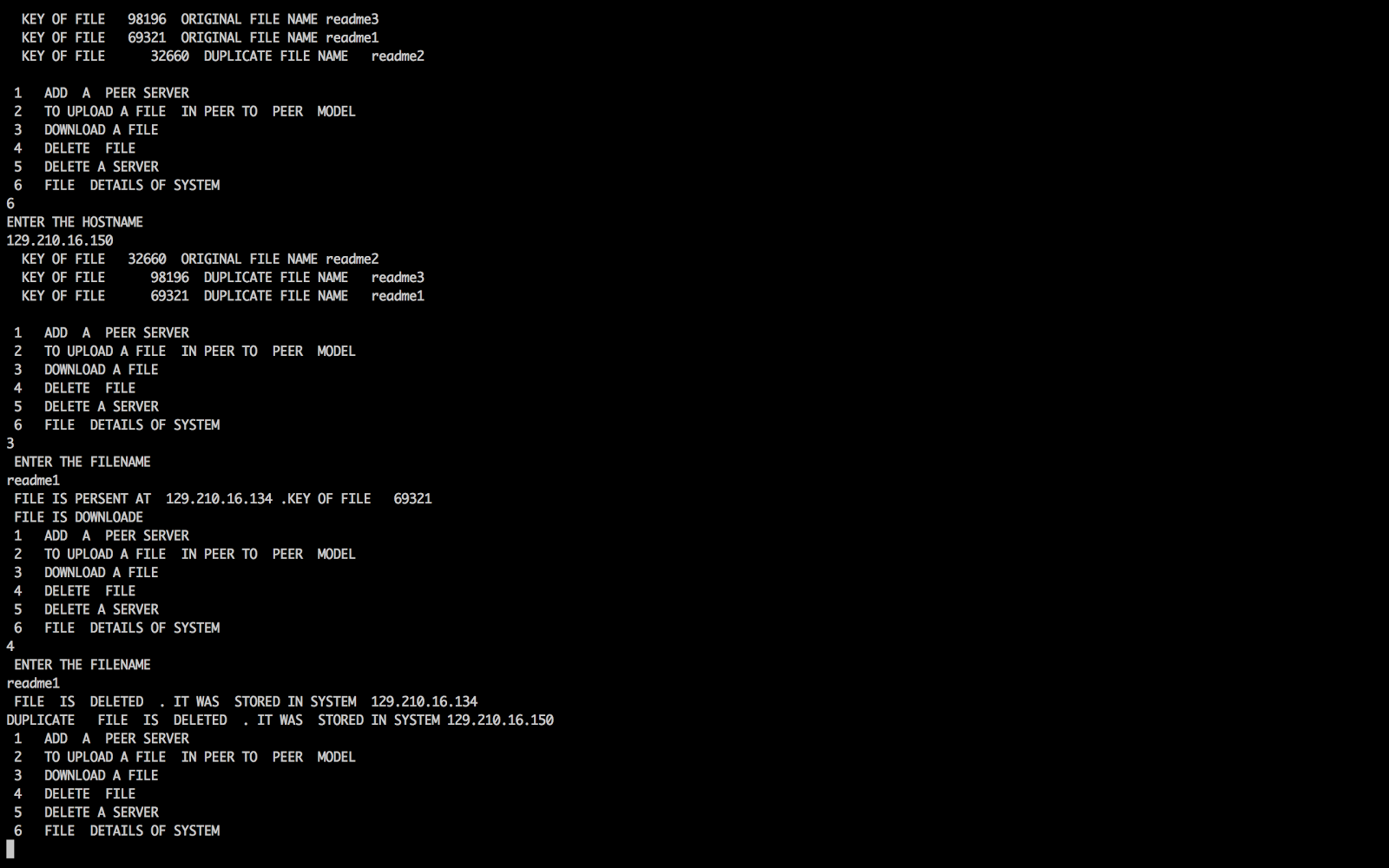
Services should run all the machines initially if you want to add the server .

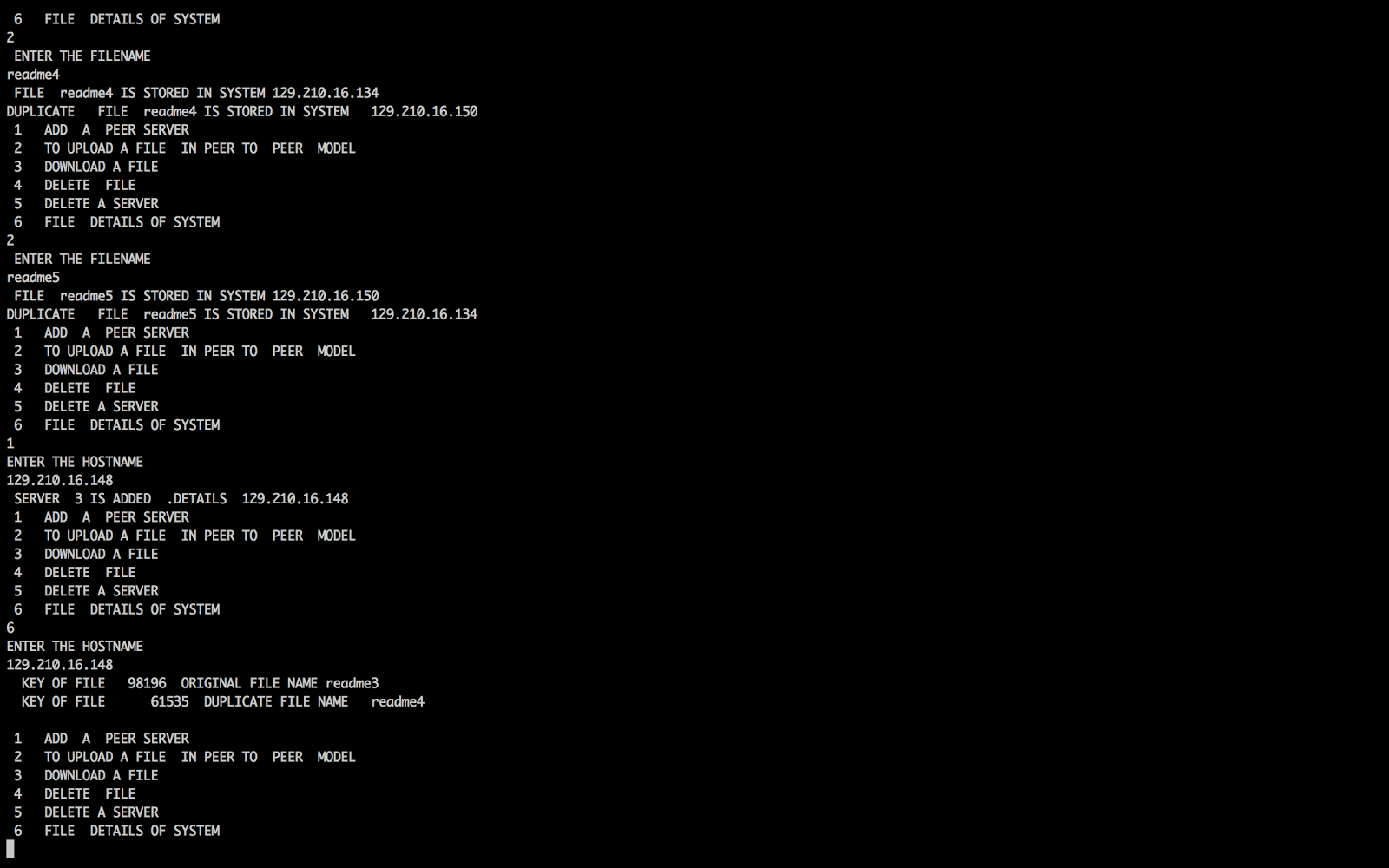




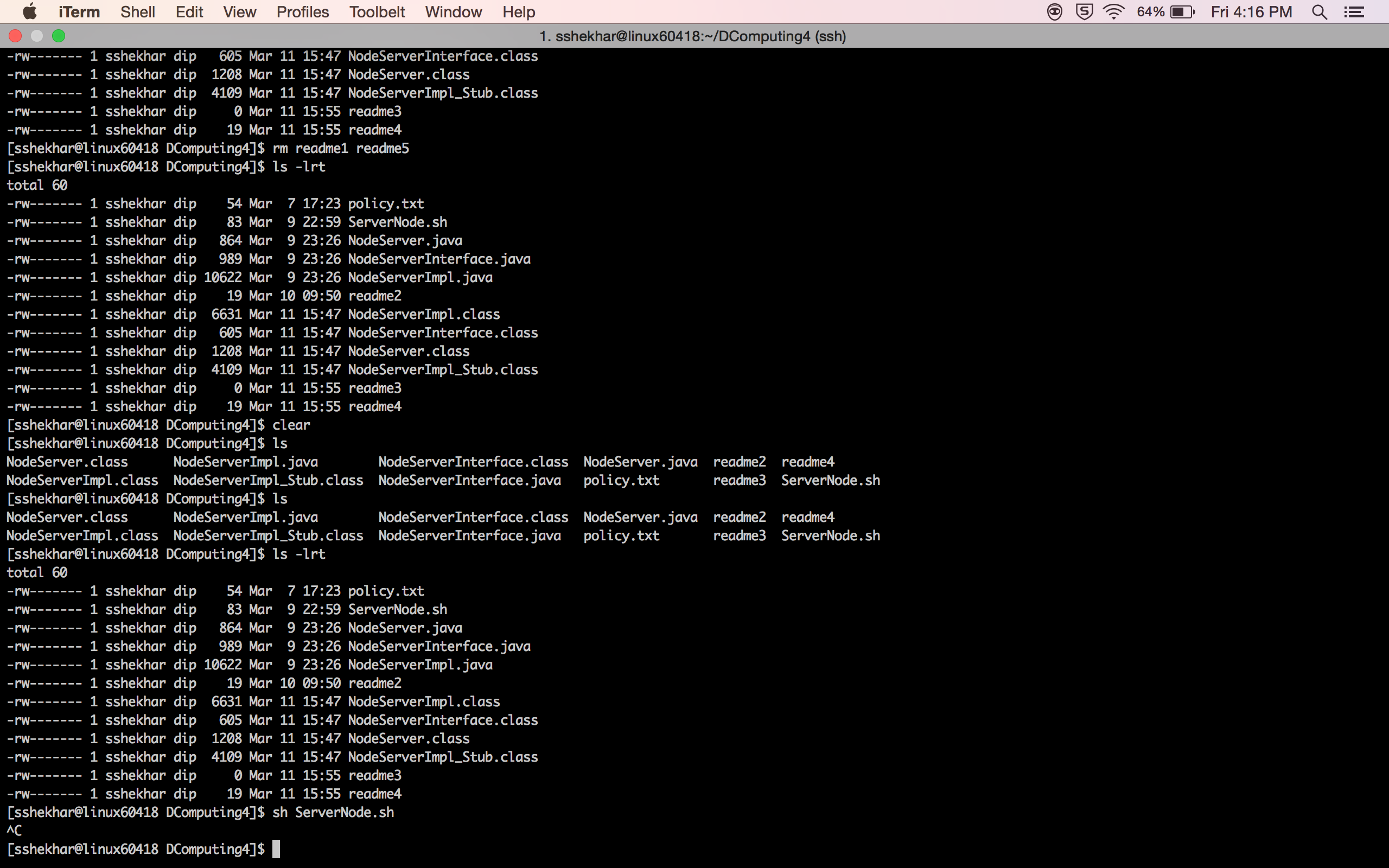




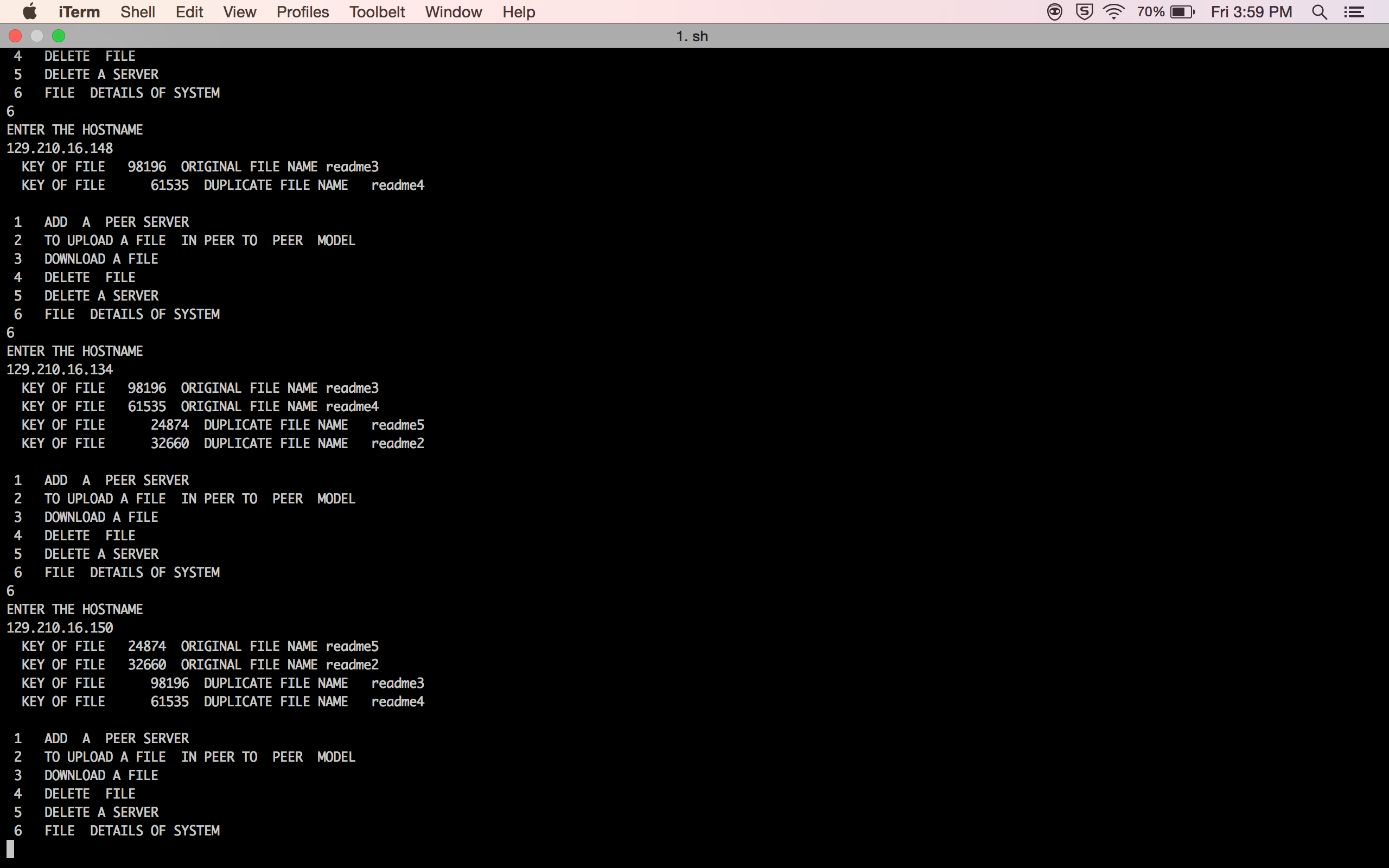


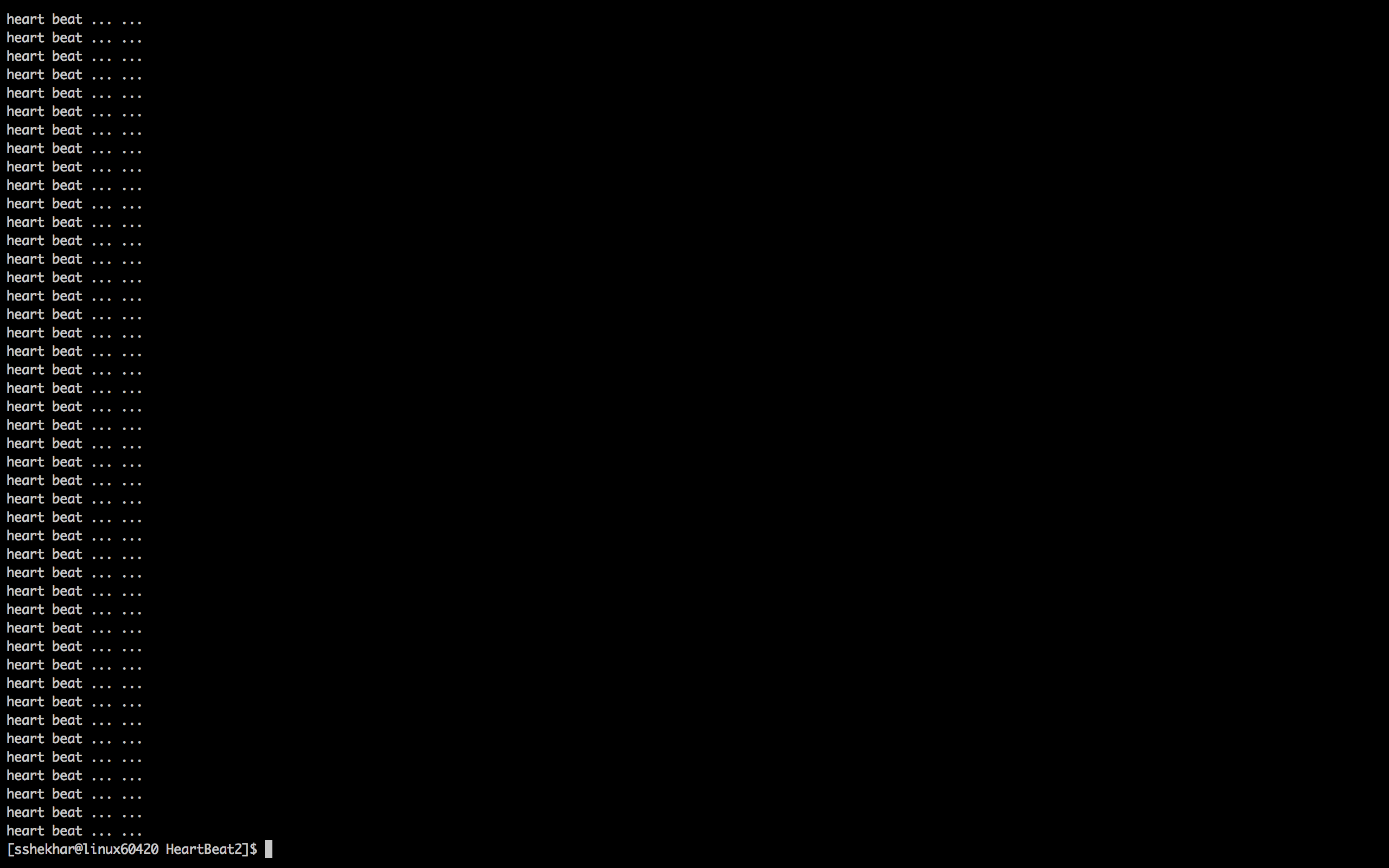






Here I stopped the rmi services of [129.210.16.148](mailto:nosshekhar@129.210.16.148) machine , Now files will get distributed in both the machines .. in next snapshots pictures of file systems of remaining nodes are given.





Server machine 129.210.16.150 machine is down . only one machine is left 129.210.16.134

So all the files will be copied in single machine. As original file .

