

# Dapster Manual

## Table of Contents

I.	Introduction
II.	System Requirements
III.	Building from Source
IV.	Running the Server
V.	Running the Peers
VI.	Troubleshooting

### I. Introduction: What is Dapster

Dapster (Dan's Napster) is a peer to peer file sharing service that works like Napster. The focus of the program is for users to share their files with each other almost directly with simple operations. In addition to the user peers, there must be a central indexing server setup somewhere. In this manual, you will learn how to build, start, and command the Dapster software to share files quickly and with ease.

For more information about how Dapster works on the inside, check the Design document as well as the provided source code in the src directory.

### II. System Requirements

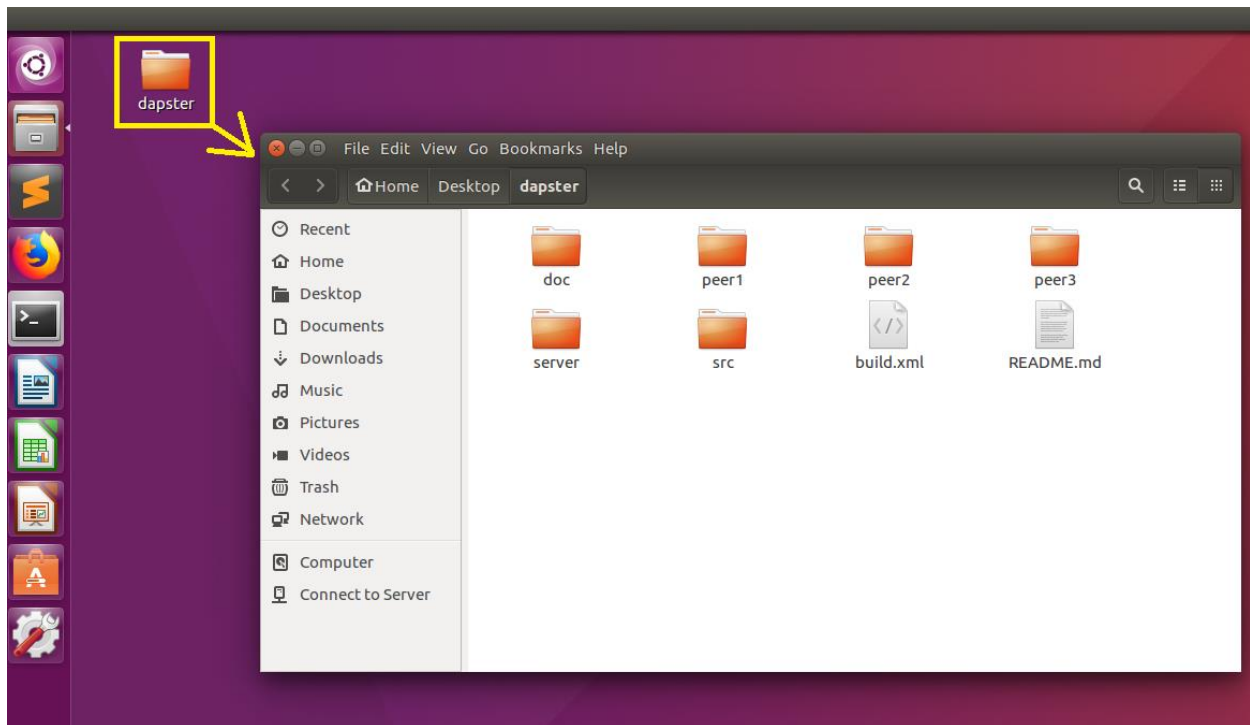
To get started and use Dapster, you must ensure that you have the following components:

- Linux (Windows and MacOS work too, but this manual assumes you are running Linux.)
- Java JDK version 8 or higher
- Apache Ant

### III. Building From Source

This software package comes with the source code, sample text files for 3 servers, and an Apache Ant build.xml file. In order to build the program, please follow these steps:

1. Make sure that the dapster directory is extracted to your desired location.



*Figure 1: The dapster directory is extracted to the Desktop.*

2. Open a Terminal and navigate to the extracted dapster directory. This can easily be done in certain version of Linux such as Ubuntu by right clicking in the Files manager and selecting "Open in Terminal".

```
Terminal File Edit View Search Terminal Help
daniel@daniel-ubuVirtualBox:~$ cd Desktop/dapster/
daniel@daniel-ubuVirtualBox:~/Desktop/dapster$
```

Figure 2: Inside the dapster directory in Terminal.

3. Final step: inside the terminal, type **ant** and press ENTER.

```
Terminal File Edit View Search Terminal Help
daniel@daniel-ubuVirtualBox:~$ cd Desktop/dapster/
daniel@daniel-ubuVirtualBox:~/Desktop/dapster$ ant
Buildfile: /home/daniel/Desktop/dapster/build.xml

compile:
  [mkdir] Created dir: /home/daniel/Desktop/dapster/build/class
  [javac] Compiling 6 source files to /home/daniel/Desktop/dapster/build/class
  [javac] Note: /home/daniel/Desktop/dapster/src/ServerStub.java uses unchecked
  or unsafe operations.
  [javac] Note: Recompile with -Xlint:unchecked for details.

jar:
  [jar] Building jar: /home/daniel/Desktop/dapster/server/Server.jar
  [jar] Building jar: /home/daniel/Desktop/dapster/peer1/Peer.jar
  [jar] Building jar: /home/daniel/Desktop/dapster/peer2/Peer.jar
  [jar] Building jar: /home/daniel/Desktop/dapster/peer3/Peer.jar

main:

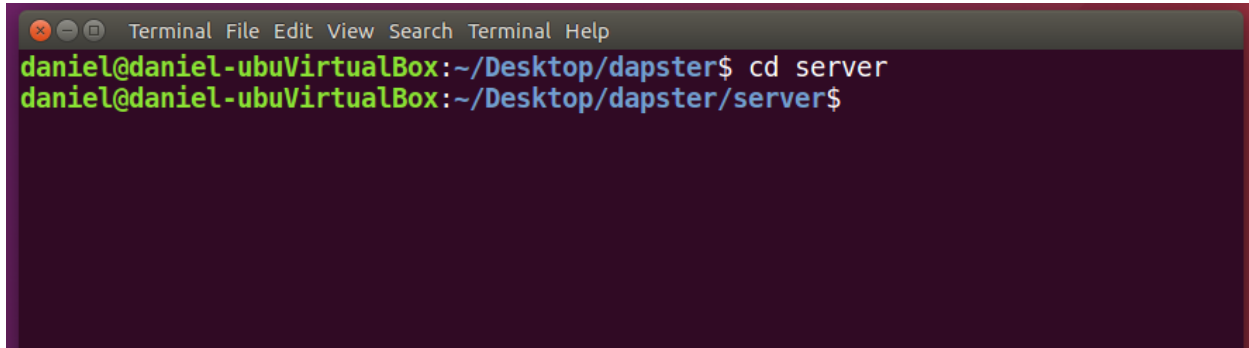
BUILD SUCCESSFUL
Total time: 1 second
daniel@daniel-ubuVirtualBox:~/Desktop/dapster$
```

Figure 3: A successful build after executing ant.

## IV. Running the Server

Once the build is completed, your installation directory becomes equipped with a Central Indexing Server as well as 3 Peers. The first step in successfully running the program is starting the server.

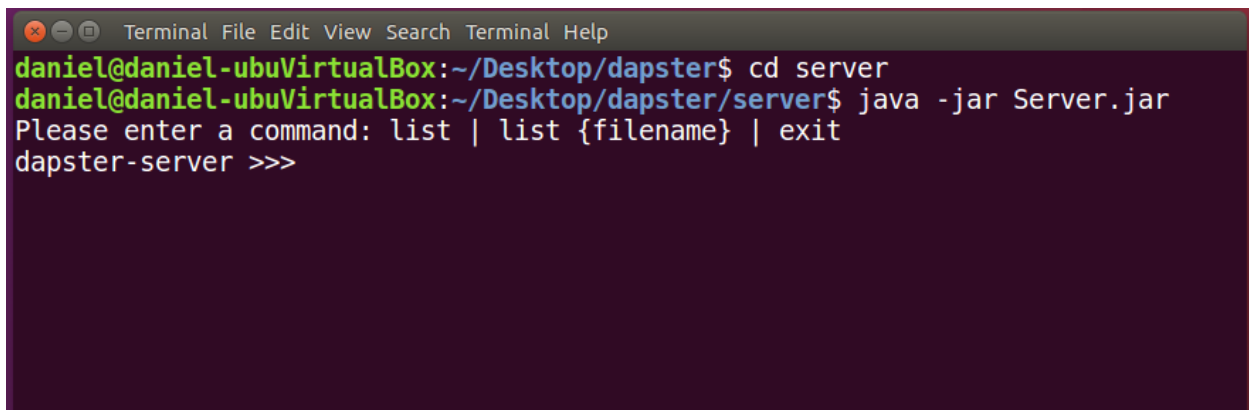
1. Change directories in your Terminal to the server directory by typing **cd server** and pressing enter.

A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'daniel@daniel-ubuVirtualBox:~/Desktop/dapster\$'. The user enters 'cd server' and the prompt changes to 'daniel@daniel-ubuVirtualBox:~/Desktop/dapster/server\$'.

```
daniel@daniel-ubuVirtualBox:~/Desktop/dapster$ cd server
daniel@daniel-ubuVirtualBox:~/Desktop/dapster/server$
```

Figure 4: The view of the terminal after clearing the screen and changing directories to the server.

2. To start running the server, type **java -jar Server.jar** and press ENTER.

A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'daniel@daniel-ubuVirtualBox:~/Desktop/dapster\$'. The user enters 'cd server' and the prompt changes to 'daniel@daniel-ubuVirtualBox:~/Desktop/dapster/server\$'. The user then enters 'java -jar Server.jar'. The terminal displays 'Please enter a command: list | list {filename} | exit' and 'dapster-server >>>'.

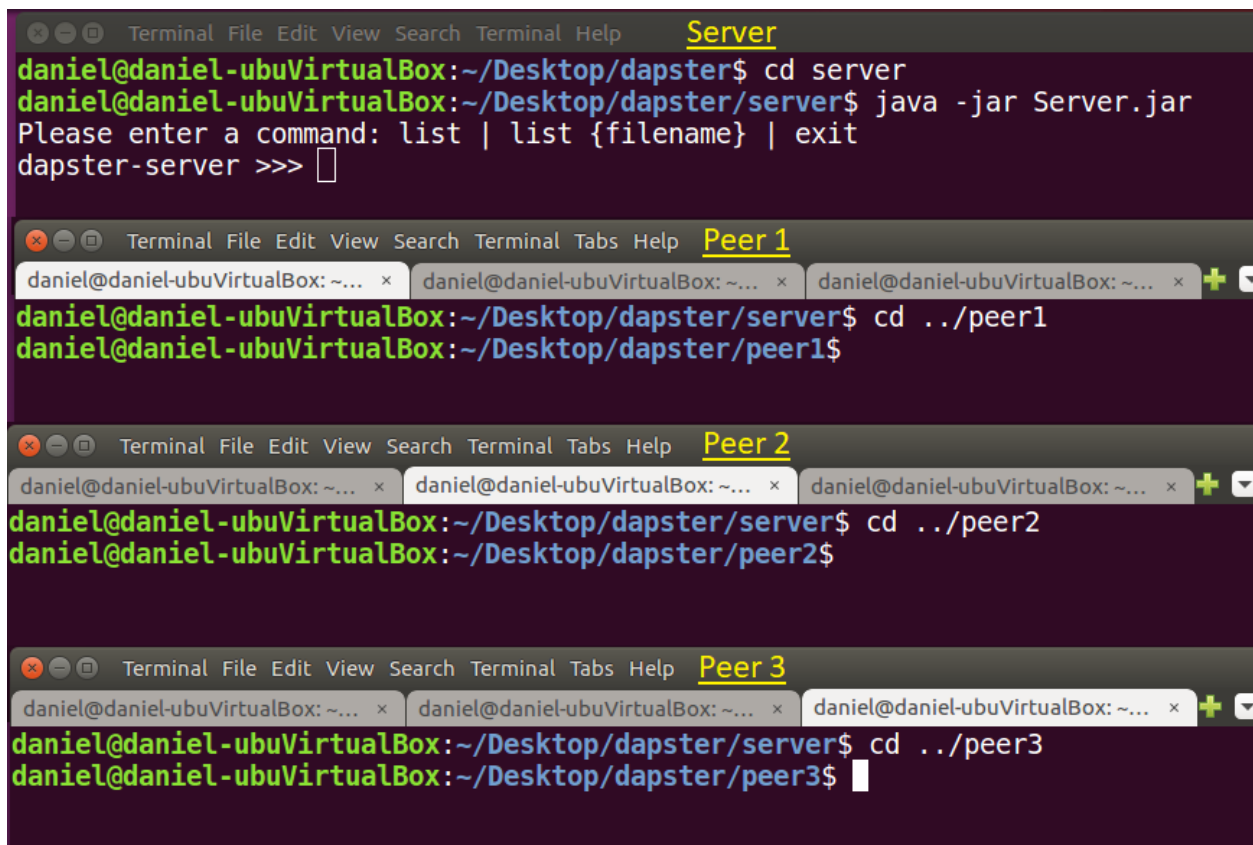
```
daniel@daniel-ubuVirtualBox:~/Desktop/dapster$ cd server
daniel@daniel-ubuVirtualBox:~/Desktop/dapster/server$ java -jar Server.jar
Please enter a command: list | list {filename} | exit
dapster-server >>>
```

3. Once the server is running, peers will be able to request services from it. In addition, you could provide the following commands to the server via the command line:
  - **list** – lists the index contents, including all files and how many peers have that file.
  - **list {filename}** – lists the number of peers that have the given file as well as the addresses of peers that have the file
  - **exit** – closes the server

## V. Running the Peers

Once the server is running, Peers can function properly. The Peer program is located in three directories: peer1, peer2, and peer3. Each peer comes equipped with its own set of 10 text files in the shared directory, ranging from 1K to 10K and name according to the peer and size. You are free to create your own files inside the shared directory or move files from another directory. This part of the manual will describe how to use the three pairs simultaneously.

1. Open three terminal tabs or separate windows. In Ubuntu's terminal, creating new tabs can easily be done by pressing **Ctrl+Shift+Tab** in the Terminal. For the first terminal, navigate to the `dapster/peer1/` directory. For the second terminal, navigate to the `dapster/peer2/` directory. For the third directory, navigate to the `dapster/peer3` directory.



The image shows three terminal windows stacked vertically. The top window, titled 'Server', shows the user 'daniel' at 'daniel-ubuVirtualBox' navigating to the server directory and running 'java -jar Server.jar'. The prompt 'Please enter a command: list | list {filename} | exit' is shown, followed by 'dapster-server >>>' and a cursor. The middle window, titled 'Peer 1', shows the user navigating from the server directory to the peer1 directory. The bottom window, titled 'Peer 2', shows the user navigating from the server directory to the peer2 directory. A third window titled 'Peer 3' is partially visible at the bottom, showing navigation to the peer3 directory. Each window has its own tab bar with the title and a '+' icon for new tabs.

```
daniel@daniel-ubuVirtualBox:~/Desktop/dapster$ cd server
daniel@daniel-ubuVirtualBox:~/Desktop/dapster/server$ java -jar Server.jar
Please enter a command: list | list {filename} | exit
dapster-server >>>

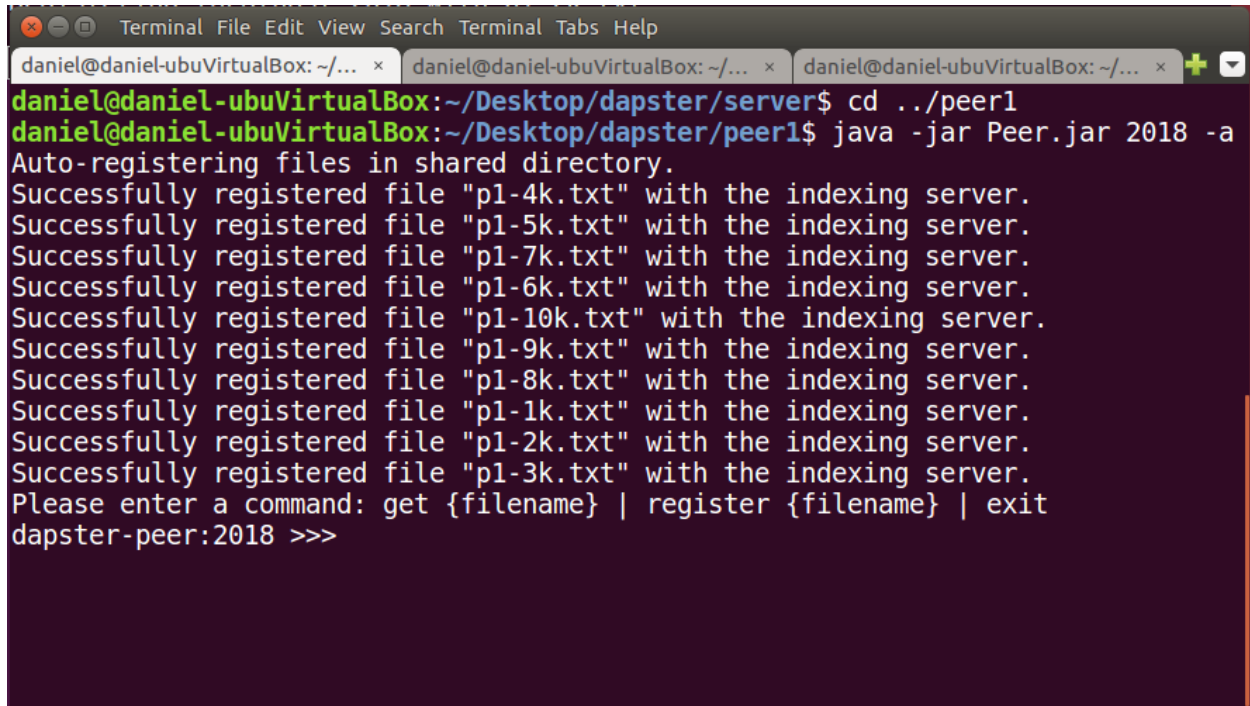
daniel@daniel-ubuVirtualBox:~/Desktop/dapster/server$ cd ../peer1
daniel@daniel-ubuVirtualBox:~/Desktop/dapster/peer1$

daniel@daniel-ubuVirtualBox:~/Desktop/dapster/server$ cd ../peer2
daniel@daniel-ubuVirtualBox:~/Desktop/dapster/peer2$

daniel@daniel-ubuVirtualBox:~/Desktop/dapster/server$ cd ../peer3
daniel@daniel-ubuVirtualBox:~/Desktop/dapster/peer3$
```

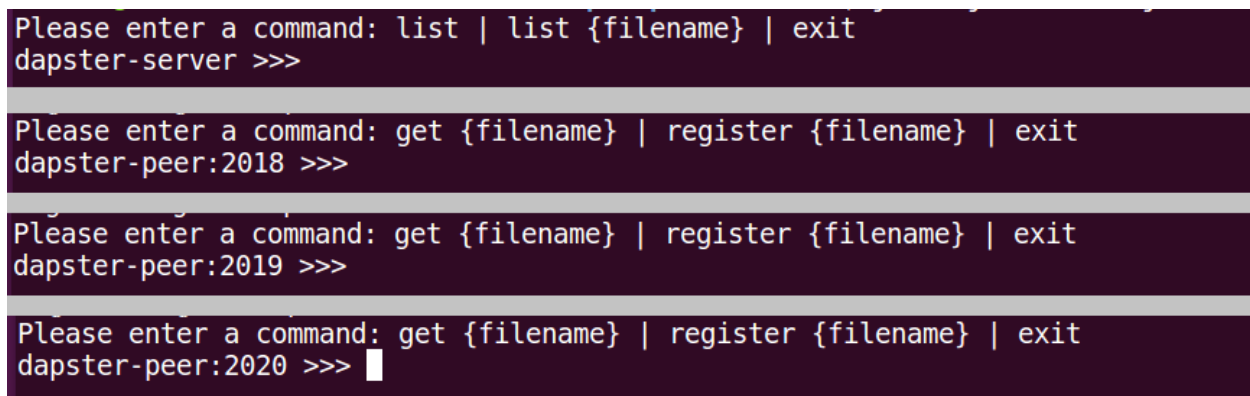
Figure 5: The Server running in its own window and three terminal tabs open in each peer directory.

2. In each of the peer terminals, start the Peer program with a different port number. In this case, the ports that are used are 2018, 2019, and 2020. Any number around these should work as well (do not use 1888, as that is the server's default port). To start the peer program in a given terminal and automatically register its files with the indexing server, type `java -jar Peer.jar PORT -a` where PORT is the unique port for that individual Peer.

A terminal window with three tabs. The active tab shows the command `cd ../peer1` followed by `java -jar Peer.jar 2018 -a`. The output shows that 12 files were successfully registered with the indexing server. The prompt is now `dapster-peer:2018 >>>`.

```
daniel@daniel-ubuVirtualBox: ~/... x daniel@daniel-ubuVirtualBox: ~/... x daniel@daniel-ubuVirtualBox: ~/... x +
daniel@daniel-ubuVirtualBox:~/Desktop/dapster/server$ cd ../peer1
daniel@daniel-ubuVirtualBox:~/Desktop/dapster/peer1$ java -jar Peer.jar 2018 -a
Auto-registering files in shared directory.
Successfully registered file "p1-4k.txt" with the indexing server.
Successfully registered file "p1-5k.txt" with the indexing server.
Successfully registered file "p1-7k.txt" with the indexing server.
Successfully registered file "p1-6k.txt" with the indexing server.
Successfully registered file "p1-10k.txt" with the indexing server.
Successfully registered file "p1-9k.txt" with the indexing server.
Successfully registered file "p1-8k.txt" with the indexing server.
Successfully registered file "p1-1k.txt" with the indexing server.
Successfully registered file "p1-2k.txt" with the indexing server.
Successfully registered file "p1-3k.txt" with the indexing server.
Please enter a command: get {filename} | register {filename} | exit
dapster-peer:2018 >>>
```

Figure 6: Starting first Peer.

A vertical stack of four terminal snippets. The first shows the server prompt `dapster-server >>>` with the command `list`. The next three show peer prompts `dapster-peer:2018 >>>`, `dapster-peer:2019 >>>`, and `dapster-peer:2020 >>>`, each with the command `get {filename} | register {filename} | exit`.

```
Please enter a command: list | list {filename} | exit
dapster-server >>>

Please enter a command: get {filename} | register {filename} | exit
dapster-peer:2018 >>>

Please enter a command: get {filename} | register {filename} | exit
dapster-peer:2019 >>>

Please enter a command: get {filename} | register {filename} | exit
dapster-peer:2020 >>> |
```

Figure 7: Server and all 3 Peers running simultaneously.

3. Once a peer is running, it will be ready to send files to other peers that request them.

Here are the available commands that you could issue a peer via the command line:

- **get {filename}** – downloads the file with the given name if a peer has it (the peer must have registered it with the indexing server). It will also automatically register the file with the server once it has been downloaded.
- **register {filename}** – registers the given file with the indexing server (will not allow registering a non-existent file)
- **test {times}** – a testing command that sends lookup requests to the server times amount of times, then reports the total and average time per request.
- **exit** – closes the peer

Examples of using these commands being used could be seen in the **Output** document.

## VI. Troubleshooting

Here are some solutions to potential problems:

**Problem:** `Java.net.BindException: Address already in use (Bind failed)`

**Solution:**

- If this happens on the Server, then you must close an application that is running on port 1888 and restart the Server.
- If this happens on a Peer, then another Peer or program is using the same port. Restart the Peer using a different port.

**Problem:** `Java.net.ConnectException: Connection refused (Connection Refused)`

**Solution:**

- This exception happens on a Peer. It means that the central Server is not running or is unreachable. It may also be that the Peer to download from is offline. To fix this, run the necessary Server or Peer.

**Problem:** `java.net.BindException: Address already in use: connect`

**Solution:**

- This exception happens when you run out of dynamic ports while testing and opening too many ports when using the test function. To fix this, wait a few minutes and check your operating system's maximum number of dynamic ports to avoid this issue.