

## Lab 3: Linux Basics and Analysis of blockchain

### Objectives:

- To get familiar with Linux and its command systems
- Analyzing various properties of blockchain through simulating bitcoin

### Introduction:

The motivation of this manual is to prepare you for the subsequent labs in which we will be using the Ubuntu Linux environment. We will start off the manual with a little bit of the history of Unix and Linux. Then, read through this lab manual and complete the instructed tasks to clarify your understanding. Finally, we will run a simulator using Ubuntu and analyze it.

### Section 1:

In this section we will learn about linux and practice some commands.

### UNIX:

UNIX is a powerful operating system designed to be a multiuser and multitasking system. The original UNIX was created by Ken Thompson in 1969 at Bell Labs. Today, the term “UNIX” does not refer to a single operating system sold by a single company. Instead, it refers to any operating system that meets certain standards.

Most large-scale computers and some desktop personal computers use an UNIX operating system, which could be a generic system or one written by the computer manufacturer. Some of the more popular UNIX operating systems include Linux, Ultrix (DEC), Irix (Silicon Graphics), and Solaris (Sun Microsystems). Mac OS X is built on BSD Unix.

Since each UNIX operating system must meet the same standards, they function similarly. Thus, after you are familiar with the use of one UNIX operating system, you will easily adapt to a different one. The major differences are usually in the administration of the system— meaning, unless you are the administrator of the system, you never have to worry about that aspect.

### Linux:

The Linux (Lynn-uks) operating system was created by Linus Torvalds in 1991 while he was a graduate student at the University of Helsinki (Finland). Torvalds created Linux as an alternative to Microsoft Windows and to provide a UNIX operating system for use on the PC. Linux is and has always been an open- source project that allows other programmers to view and modify the source code. Today, hundreds of programmers work on Linux—mostly in their spare time—under the direction of Torvalds.

The Linux operating system is very popular today due in part to its availability and open source status. There are a number of Linux distributions available from different companies and groups such as Red Hat, Fedora, Ubuntu, Slackware, SuSe, and Corel. All of these use the same

Linux operating system.

The major differences between the distributions are the services provided and the various applications included with the Linux distribution. Linux is very powerful and is easy to learn and use. All major distributions provide a graphical user interface frontend that will be familiar to Microsoft and Macintosh users.

In our lab we will be using Ubuntu Linux Distribution.

## Terminal:

Users commonly interact with a UNIX system via a text-based command-line interface. In a terminal, commands are entered at a prompt and results are displayed. Numerous commands are provided for file and directory manipulation, program execution, and file processing. Before you can begin working with UNIX commands, you need a terminal window, which runs a program called a shell. The shell provides an interface between you and the operating system. The terminal can be launched using the Menu, or typing Terminal into the application area or using the Ctrl + Alt + T shortcut. The terminal window on your desktop should contain some characters that look like:

```
username@lisp:~ $
```

This is called the prompt. The prompt gives you information about the account and machine being used and the current directory you're in. For example, the prompt above is for someone whose username is **username** and who is using the computer named **lisp** and that the user is in his **home directory**, as indicated by the "**~**", the shortcut for the home directory.

The prompt indicates that UNIX is waiting for (or "prompting") you to type something. Whenever you type something after a UNIX prompt, UNIX tries to understand it as a command. If you type a command that UNIX understands, UNIX carries out the command. Otherwise UNIX displays a message indicating that the command was not recognizable.

## Linux Manual:

Unix documentation is traditionally in the form of a Unix Manual, which consists of a set of manual pages, or simply man pages, organized into nine sections. Section one of the manual is for user commands, section two for system calls, section three for higher-level API calls and so on. Sometimes you will see commands (and API functions) written as name(n). This notation specifies a name and a manual section. For example, tty(1) refers to the user command `tty`, whereas tty(4) refers to the device driver named `tty`.

You read man pages using the `man` command. The `man` command itself has a man page, which you read by issuing the command `man man`.

- Type `man man` in your terminal and try to understand what is written

To work in linux you should be comfortable reading man pages, and referring to the man pages

must become second nature. Any time you wonder how a command works, read the man pages. If you need to know what format a file has, read the man pages. If you don't have anything else to do, read a man page; you might just learn something.

Man pages are divided into named sections such as "SYNOPSIS", "DESCRIPTION", "EXAMPLES" and "FILES". If you are familiar with the more common sections of man pages you can find information a lot faster than by trying to read the whole thing from beginning to end. The man page for man itself lists some of the common sections and conventions.

## Linux File System:

Understanding how files and directories are organized and can be manipulated is vital when using or managing a Linux system. All files and directories in Linux are organized in a single tree, regardless of what physical devices are involved (unlike Microsoft Windows, where individual devices typically form separate trees).

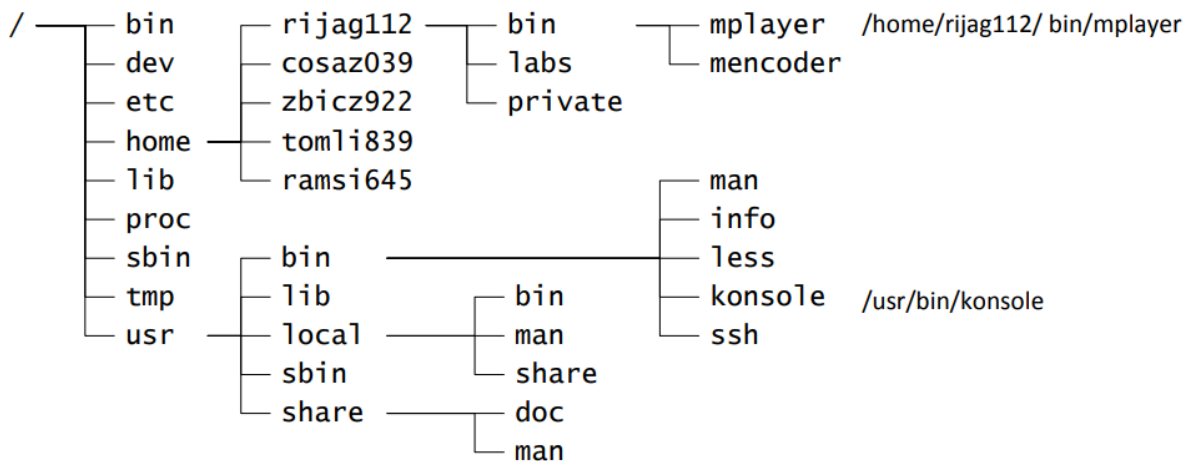
The basic unit of storage in UNIX is a file. A file may contain many kinds of information, including a Python script, an HTML document, a research paper, an image, or an executable program. Files are organized in a hierarchical system of directories. A directory may contain files and other directories. The directory at the "top" of the file system, in which all other directories are located, is called the root directory.

The root of the tree is called /, and is known as the root directory or simply the root. The root contains a number of directories, most of which are standard on Linux systems. The following top-level directories are particularly important:

Directory	Purpose
bin	Commands (binaries) needed at startup. Every Unix command is a separate executable binary file. Commands that are fundamental to operation, and may be needed while the system is starting, are stored in this directory. Other commands go in the /usr directory.
dev	Interfaces to hardware and logical devices. Hardware and logical devices are represented by device nodes: special files that are stored in this directory.
etc	Configuration files. The /etc directory holds most of the configuration of a system. In many Linux systems, /etc has a subdirectory for each installed software package.
home	Home directories. User's home directories are subdirectories of /home.
sbin	Administrative commands. The commands in /sbin typically require administrative privileges or are of interest only to system administrators. Commands that are needed when the system is starting go in /sbin. Others go in /usr/sbin.

temp	Temporary (non-persistent) files. The /tmp directory is typically implemented in main memory. Data stored here is lost when the system reboots. Many applications use /tmp for storing temporary files (others use /var).
usr	The bulk of the system, including commands and data not needed at startup. The usr subdirectory should only contain files that can be shared between a number of different computers, so it should contain no configuration data that is unique to a particular system.

The figure below shows part of a Unix system:



## File and path names:

There are two ways to reference a file in Unix: using a relative path name or using an absolute path name. An absolute path name always begins with a / and names every directory on the path from the root to the file in question. For example, in the figure above, the console file has the absolute path name /usr/bin/konsole. A relative path names a path relative to the current working directory. The current working directory is set using the cd command. For example, if the current working directory is /usr, then the konsole file could be referenced with the name bin/konsole. Note that there is no leading /. If the current working directory were /usr/share, then konsole could be referenced with ../bin/konsole. The special name “..” is used to reference the directory above the current working directory.

**Checkpoint 1: In the example above name at least one relative path indicating ssh if:**

- The current working directory is /usr/bin.**
- The current working directory is /usr/local/bin**

## File system permissions:

Like most operating systems Linux has permissions on files and directories that grant individual users or groups of users rights on the files and folders.

In Linux, permissions are divided into three groups: “user”, “group” and “other”. User permissions apply to the owner of a file or directory; group permissions to the members of the file’s (or directory’s) group; other permissions apply to everyone else.

Every group contains three main permissions: read, write and execute, and each is represented as one bit in an integer. The read (r) bit grants permission to read the contents of a file or directory; the write (w) bit grants permission to write to the file or create files in a directory, and the execute (x) bit grants permission to execute a file as a program. On directories the execute bit grants permission to traverse the directory (i.e. set it as the working directory).

There are other permission bits as well. The most important of these are the setuid and setgid bits (in the user and group permission groups, respectively). When a program with the setuid bit set is run, it is run as the owner of the file, not the user who started the program. The setgid bit works the same, but for groups

To list the permissions of a file or directory, use the ls command with the -l option (to enable long file listing; see the man page for ls). You can see something like this:

```
ls -l foobar
-rwxr-xr-- 1 john users 64 May 26 09: 55 foobar
```

Each group of permissions is represented by three characters in the leftmost column of the listing. The very first character indicates the type of the file, and is not related to permissions. The next three characters (in this case rwx) represent user permissions. The following three (in this case r-x) represent group permissions and the final three represent permissions for others (in this case r--).

The owner and group of the file are given by the third and fourth column, respectively (user john and group users in this example).

In this example the owner, “john”, is allowed to read, write and execute the file (rwx). Users belonging to the group “users” are allowed to read and execute the file (r-x), but cannot write to it. All other users are allowed to read foobar (r--), but not write or execute it.

The first character, the type field, indicates the file type. In the example above the file type is “-”, which indicates a regular file. Other file types include: d for directory, l (lower case ell) for symbolic links, s for Unix domain socket, p for named pipe, c for character device file and b for block device file.

**Checkpoint 2:** Use the ls command to get the file list and then use the -l option for any two files and examine what you can see.

## Manipulating access right:

The chmod and chown commands are used to manipulate permissions.

chmod is used to manipulate permissions. Permissions can be specified using either “long” format or a numeric mode (all permission bits together are called the file’s mode). The long format takes a string of permission values (r, w or x) together with a plus or minus sign. For example, to prevent any user from changing the file foobar we would do as follows to disable write permission, then verify that the change has taken place:

```
chmod -w foobar
```

Now, we can check the updated permission again:

```
ls -l foobar  
-r -xr -xr -x 1 john users 81 May 26 10: 43 foobar
```

In numeric mode, each permission is treated as a single bit value. The read permission has value 4, write value 2 and execute value 1. The mode is a three character octal string where the first digit contains the sum of the user permissions, the second the sum of the group permissions and the third the sum of the others permissions. For example, to set the permission string “-rwxrw-r--” (user may do anything, group may read or write, but not execute and all others may read) for a file, you would calculate the mode as follows:

- User:  $4 + 2 + 1 = 7$  (rwx)
- Group:  $4 + 2 = 6$  (rw-)
- Others:  $4 = 4$  (r--)

Together with chmod the string “764” can then be used to set the file permissions.

```
chmod 764 foobar
```

Numeric combinations are generally quicker to work with once you learn them, especially when making more complicated changes to files and directories. Therefore, you are encouraged to use them. It is useful to learn a few common modes by hearts:

755	Full rights to user, execute and read rights to others. Typically used for executables.
644	Read and write rights to user, read to others. Typically used for regular files.
777	Read, write and execute rights to everybody. Rarely used.

The `chown` is used to change the owner and group for a file. To change the user from “john” to “mike” and the group from “users” to “wheel” issue

```
chown mike: wheel foobar
```

Note that some Unix systems do not support changing the group with `chown`. On these systems, use `chgrp` to change the file's group. Changing the owner of a file can only be done by privileged users such as `root`. Unprivileged users can change the group of a file to any group they are a member of. Privileged users can alter the group arbitrarily

**Checkpoint 3: What do the following numeric file modes represent:**

- a) 666
- b) 770
- c) 640
- d) 444

## Some regularly used linux commands:

1. `man`: Access manual pages for all Linux commands
2. `ls`: The most frequently used command in Linux to list directories
3. `ls -a`: show the list of files including the hidden files.
4. `pwd`: Print working directory command in Linux
5. `cd`: Linux command to navigate through directories
6. `mkdir`: Command used to create directories in Linux
7. `rm`: Delete files or directories
8. `touch`: Create blank/empty files
9. `cat`: Display file contents on the terminal
10. `clear`: Clear the terminal display
11. `echo`: Print any text that follows the command
12. `<command> -h` or `<command> -help`: shows the available command details
13. `whoami`: Get the active username
14. `export`: Export environment variables in Linux
15. `ps`: Display active processes
16. `kill` and `killall`: Kill active processes by process ID or name
17. `chmod`: Command to change file permissions
18. `chown`: Command for granting ownership of files or folders
19. `apt`, `rpm`: Package managers depending on the distribution
20. `apt-get`: Can be used to install, update, upgrade packages
21. `sudo`: Command to provide administrative privileges in Linux
22. `useradd` and `usermod`: Add new user or change existing users data
23. `passwd`: Create or update passwords for existing users

## Section 2:

In this section we will use a blockchain simulator. Using the simulator, we will simulate and analyze results of bitcoin transactions. From the generated results, we generate graphs and try to understand how different parameters affect the results.

### BlockSim:

[BlockSim](#) is an open source blockchain simulator. It can be used to analyze and test various aspects of blockchain. You can read the official documentation of BlockSim from [here](#).

### Setting the environment:

To run the simulation, it is recommended that you have a python version 3 or above installed in your machine. Ubuntu22.x comes with pre-built python version 3. Therefore, it should be installed on the computer. You can check your python version by issuing the command:

```
yeasin49@yeasin49-X556URK:~$ python3 --version
Python 3.10.6
```

You should see a version starting from 3.x.x. Additionally, you should have some packages. To install them, you also need pip3 installed on your computer. Run the command below and you should see a version for pip3.

```
yeasin49@yeasin49-X556URK:~$ pip3 --version
pip 23.0 from /home/yeasin49/.local/lib/python3.10/site-packages/pip (python 3.10)
```

If you do not have it installed. Run the following command:

```
sudo apt install -y python3-pip
```

After installing it, check the version of pip3 again. This time you should see a version in your terminal.

If everything is fine, download the following packages:

- pandas

```
pip3 install pandas
```

- numpy

```
pip3 install numpy
```

- sklearn

```
pip3 install sklearn
```

If the above command gives error try this: `pip3 install scikit-learn`

- xlswriter

```
pip3 install XlsxWriter
```



Finally, you are almost ready to use the [BlockSim](#). Download the project files or you can clone the repository using the command:

```
git clone https://github.com/maher243/BlockSim.git
```

Now, open the BlockSim folder in your vscode. After successfully opening the folder in vscode, open the terminal from vscode and you should see the default path of your terminal is "BlockSim" like below:



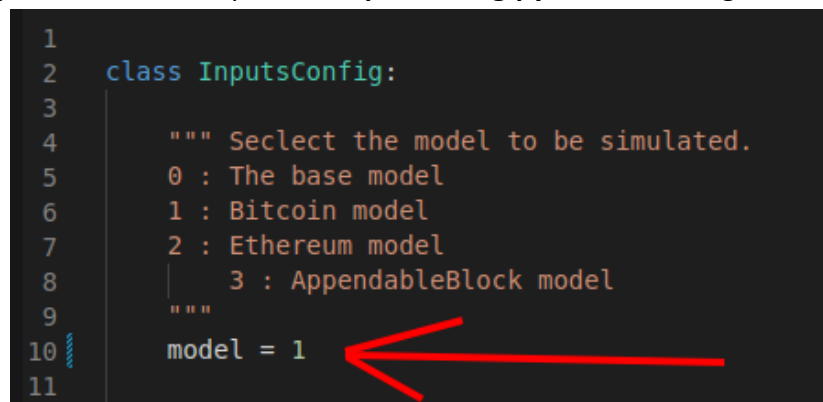
```
yeasin49@yeasin49-X556URK:~/Documents/Lab03/BlockSim$
```

## Running the simulator:

BlockSim currently simulates 3 models of blockchain. Base Model, Bitcoin model and Ethereum Model. As a user, you have the ability to configure many aspects of simulation. You should see a file named **InputsConfig.py**. This is the file, where we can configure many inputs as per our needs for the simulation.

In this lab, we will simulate the bitcoin model and observe how different input parameters affect the final results.

To run the simulation in the bitcoin model, we need to change the model value from the **InputsConfig.py** file. Therefore, open the **InputsConfig.py** file and change the model = 1.



```
1
2 class InputsConfig:
3
4     """ Select the model to be simulated.
5     0 : The base model
6     1 : Bitcoin model
7     2 : Ethereum model
8     3 : AppendableBlock model
9     """
10    model = 1
11
```

If you run the simulation now, it will simulate the bitcoin model. However, we will configure more inputs to analyze some aspects of bitcoin. In the **InputsConfig.py** file you will see an **if** block executes code for model = 1( **if model = 1** ). We will configure inputs from this block.

## Experiment-1

In this experimental simulation we will try to understand what happens when the total number of miners increases in the network. By default, you will find 3 miners(NODES) in the simulation configuration. To start the experiment,

1. First change the **Runs** to 10 like below:

```

41 ''' Input configurations for Bitcoin model '''
42 if model == 1:
43     ''' Block Parameters '''
44     Binterval = 600 # Average time (in seconds)for creating a block in the blockchain
45     Bsize = 1.0 # The block size in MB
46     Bdelay = 0.42 # average block propagation delay in seconds, #Ref: https://bitslog.wordpress.com/201
47     Breward = 12.5 # Reward for mining a block
48
49     ''' Transaction Parameters '''
50     hasTrans = True # True/False to enable/disable transactions in the simulator
51     Ttechnique = "Light" # Full/Light to specify the way of modelling transactions
52     Tn = 10 # The rate of the number of transactions to be created per second
53     # The average transaction propagation delay in seconds (Only if Full technique is used)
54     Tdelay = 5.1
55     Tfee = 0.000062 # The average transaction fee
56     Tsize = 0.000546 # The average transaction size in MB
57
58     ''' Node Parameters '''
59     Nn = 3 # the total number of nodes in the network
60     NODES = []
61     from Models.Bitcoin.Node import Node
62     # here as an example we define three nodes by assigning a unique id for each one + % of hash (comput
63     NODES = [
64         Node(id=0, hashPower=50), Node(
65             id=1, hashPower=20), Node(id=2, hashPower=30)]
66
67     ''' Simulation Parameters '''
68     simTime = 10000 # the simulation length (in seconds)
69     Runs = 10 # Number of simulation runs
70
71 ''' Input configurations for Ethereum model '''

```

- Now, run the simulation using the following command:

```
python3 Main.py
```

You may see a warning in the terminal window like below which can be ignored for the time being:

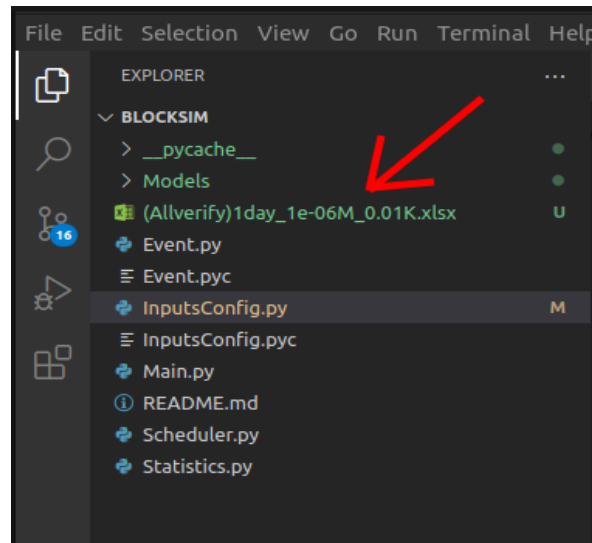
```

• yeasin49@yeasin49-X556URK:~/Documents/Lab03/BlockSim$ python3 Main.py
/home/yeasin49/Documents/Lab03/BlockSim/Statistics.py:96: FutureWarning: save is not part of the
public API, usage can give unexpected results and will be removed in a future version
    writer.save()
/home/yeasin49/Documents/Lab03/BlockSim/Statistics.py:96: FutureWarning: save is not part of the
public API, usage can give unexpected results and will be removed in a future version
    writer.save()
/home/yeasin49/Documents/Lab03/BlockSim/Statistics.py:96: FutureWarning: save is not part of the
public API, usage can give unexpected results and will be removed in a future version
    writer.save()
/home/yeasin49/Documents/Lab03/BlockSim/Statistics.py:96: FutureWarning: save is not part of the
public API, usage can give unexpected results and will be removed in a future version
    writer.save()

```

However, If you want you can remove this warning. Also if you are getting errors then you need to fix it. To do this, open **Satistics.py** file and In line 96, you will see something like **writer.save()**. Renaming it with **writer.close()** will remove the warning message.

Additionally you will see the simulation will generate a **xlsx** file which will contain the results of the simulation.



Open the **xlsx** file and check the profit tab. You will see the data of 3 miners. Check the row with non-zero values.

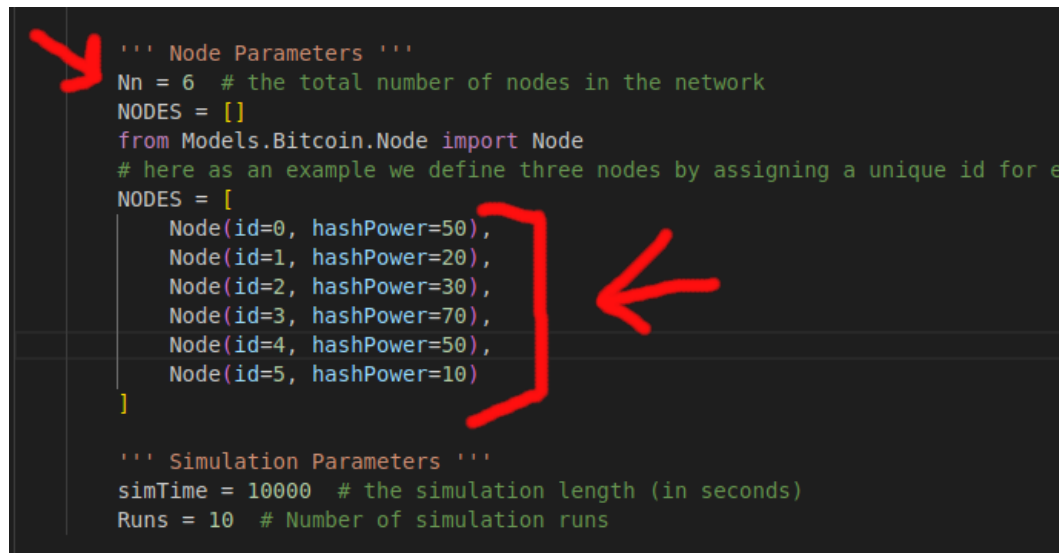
	A	B	C	D	E	F	G	H
1		Miner ID	% Hash Power	# Mined Blocks	% of main blocks	# Uncle Blocks	% of uncles	Profit (in ETH)
2	0	0	50	9	45	0	0	114.7470196125
3	1	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0
5	3	0	0	0	0	0	0	0
6	4	0	0	0	0	0	0	0
7	5	0	0	0	0	0	0	0
8	6	0	0	0	0	0	0	0
9	7	0	0	0	0	0	0	0
10	8	0	0	0	0	0	0	0
11	9	0	0	0	0	0	0	0
12	10	1	20	3	15	0	0	38.24488689317
13	11	0	0	0	0	0	0	0
14	12	0	0	0	0	0	0	0
15	13	0	0	0	0	0	0	0
16	14	0	0	0	0	0	0	0
17	15	0	0	0	0	0	0	0
18	16	0	0	0	0	0	0	0
19	17	0	0	0	0	0	0	0
20	18	0	0	0	0	0	0	0
21	19	0	0	0	0	0	0	0
22	20	2	30	8	40	0	0	101.9819327635
23	21	0	0	0	0	0	0	0
24	22	0	0	0	0	0	0	0
25	23	0	0	0	0	0	0	0
26	24	0	0	0	0	0	0	0
27	25	0	0	0	0	0	0	0
28	26	0	0	0	0	0	0	0
29	27	0	0	0	0	0	0	0
30	28	0	0	0	0	0	0	0
31	29	0	0	0	0	0	0	0
32								
33								
34								
35								
36								
37								
38								
39								
40								

<

3. Now, we will change the configuration value and run the simulation again. Before that,

make sure to rename this file, or save the data of this file. Else, when a new simulation gets started, the file data will get replaced.

4. Now, change total number of node count **Nn** to 6 and add 3 more **Node** having a unique ID inside **NODES** array like image below:



```
''' Node Parameters '''
Nn = 6 # the total number of nodes in the network
NODES = []
from Models.Bitcoin.Node import Node
# here as an example we define three nodes by assigning a unique id for e
NODES = [
    Node(id=0, hashPower=50),
    Node(id=1, hashPower=20),
    Node(id=2, hashPower=30),
    Node(id=3, hashPower=70),
    Node(id=4, hashPower=50),
    Node(id=5, hashPower=10)
]

''' Simulation Parameters '''
simTime = 10000 # the simulation length (in seconds)
Runs = 10 # Number of simulation runs
```

5. Now, run the simulator and rename the xlsx file to “run-2” and save the data.
6. Now, try to simulate multiple times as we did till now by changing the node count **Nn** to 9, 12, 15 by adding 3 more **Nodes** in each simulation. You must ensure the backup of xlsx file data in each simulation.

**Checkpoint 4:** Now, draw a graph illustrating the relationship between miner count and % of **Mined blocks** of an individual miner( Either for miner 0, 1 or 2 ) and answer the following:

- I. What happens to the percentage of mined blocks of an individual miner when total miner count increases ?
- II. What will probably happen to the profit of the miner when miner count remains the same ? (Answer it from your understanding)

## Experiment-2

In this experiment, we will discover the effects of block interval to the whole network.

To do this, change the **BInterval** value to 1, 5, 10, 15, 20, 25 of the config file and notice what happens to the **stale rate** in the **simOut** tab of the generated report of simulation.

**Note:** One of the concerning results of a higher stale rate is, it affects the security of the blockchain network.

**Checkpoint 5:**

- I. Draw a graph illustrating the relationship between block interval( **BInterval** ) and stale rate.
- II. What happens to the stale rate when the **BInterval** increases ?

**Submission Link:** [CSE446 Lab03 Submission](#)

**References**

1. Read more about the simulator from :  
<https://www.frontiersin.org/articles/10.3389/fbloc.2020.00028/full>