PRIVATE GIT SERVER - USING SSH PROTOCOL

By

Mohammed Suhail Roushan Ali

CS.CODE.IN

2022

ABSTRACT

Secure Shell(SSH) is basically a cryptographic protocol or an interface that can implement network service and communication with the needs of a remote computer. It authorizes users to conduct network communication and other utility on unsecured networks. SSH was firstly planned to toil on a client/server architecture and client securely authenticate and send encrypted data to be implemented on to the server. SSH mostly uses AES, IDEA and BLOWFISH.

The Secure Shell (SSH) is a protocol for secure remote login and other secure network services over an insecure network. This document describes the SSH transport layer protocol, which typically runs on top of TCP/IP. The protocol can be used as a basis for a number of secure network services. It provides strong encryption, server authentication, and integrity protection. It may also provide compression. Key exchange method, public key algorithm, symmetric encryption algorithm, message authentication algorithm, and hash algorithm are all negotiated. This document also describes the Diffie-Hellman key exchange method and the minimal set of algorithms that are needed to implement the SSH transport layer protocol.

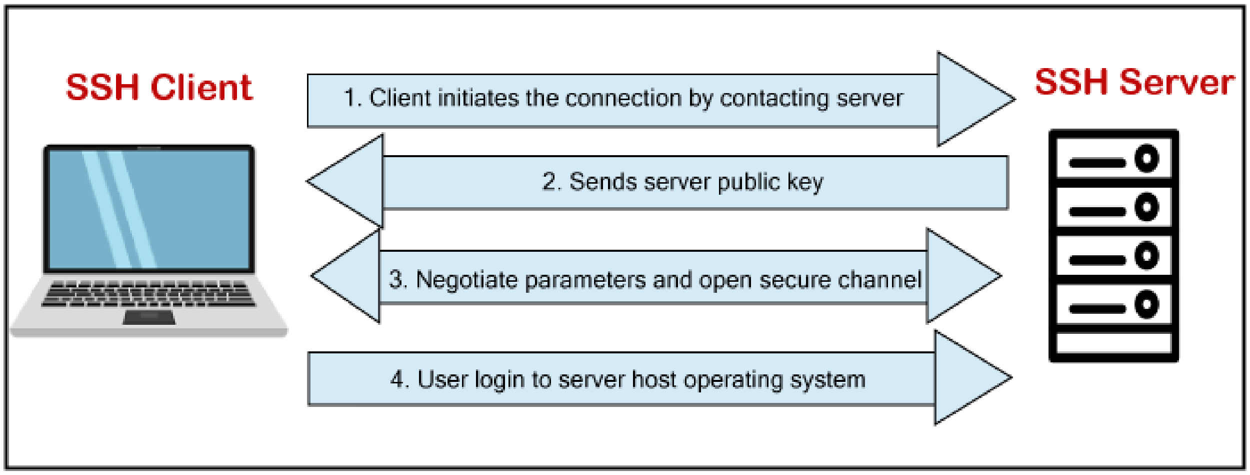
Keywords: Secure Shell, Protocol, Encryption, Communication, Telnet

TABLE OF CONTENTS

1. What is SSH

SSH, the Secure Shell, is a popular, powerful, software-based approach to network security. Whenever data is sent by a computer to the network, SSH automatically encrypts it. When the data reaches its intended recipient, SSH automatically decrypts (unscrambles) it. The result is transparent encryption: users can work normally, unaware that their communications are safely encrypted on the network. In addition, SSH uses modern, secure encryption algorithms and is effective enough to be found within mission-critical applications at major corporations. SSH has a client/server architecture,

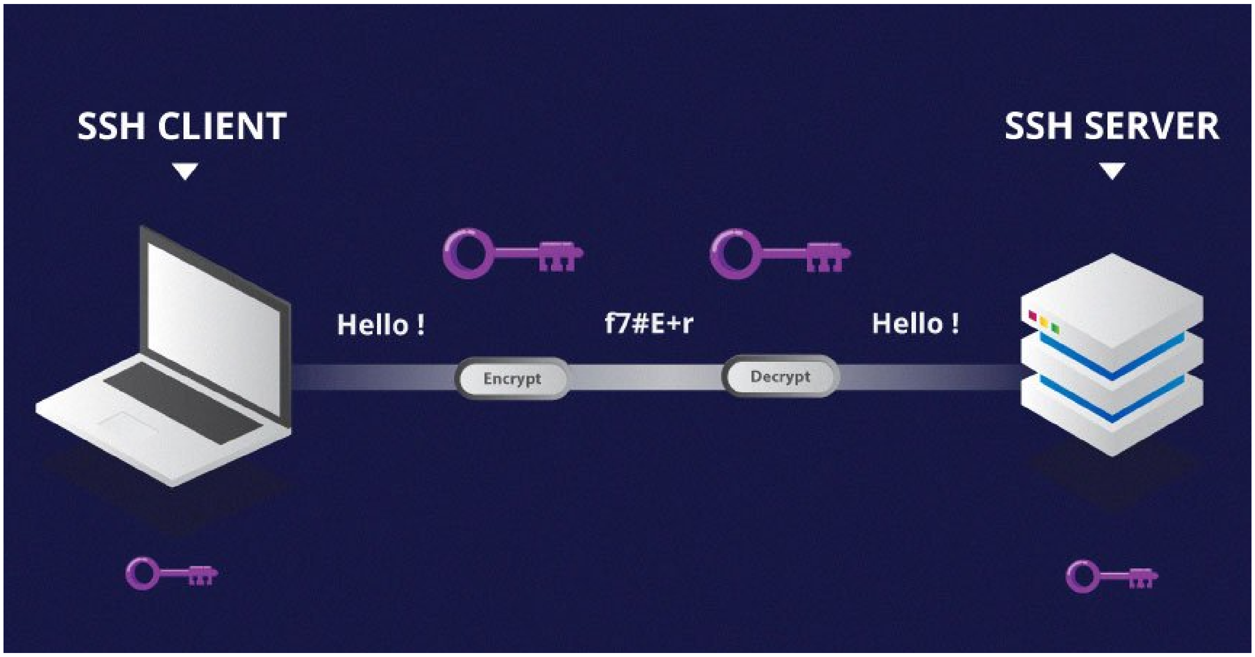
An SSH server program, typically installed and run by a system administrator, accepts or rejects incoming connections to its host computer. Users then run SSH client programs, typically on other computers, to make requests of the SSH server, such as "Please log me in," "Please send me a file," or "Please execute this command." All communications between clients and servers are securely encrypted and protected from modification


SSH Client
SSH Server
1. Client initiates the connection by contacting server
2. Sends server public key
3. Negotiate parameters and open secure channel
4. User login to server host operating system

An SSH-based product might include clients, servers, or both. Unix products generally contain both clients and servers; those on other platforms are usually just clients, though Windows-based servers are beginning to appear. If you're a

Unix user, think of SSH as a secure form of the Unix r-commands: rsh (remote shell), rlogin (remote login), and rcp (remote copy).

In fact, **the original SSH for Unix includes the similarly named commands ssh, scp, and slogin as secure, drop-in replacements for the r-commands.** Yes, you can finally get rid of those insecure .rhosts and hosts.equiv files! (Though SSH can work with them as well, if you like.) If you're still using the r-commands, switch to SSH immediately: the learning curve is small, and security is far better



2. SSH Protocol

SSH is a protocol, not a product. It is a specification of how to conduct secure communication over a network.

The SSH protocol covers **authentication, encryption, and the integrity of data** transmitted over a network,

Authentication Reliably determines someone's identity. If you try to log into an account on a remote computer, SSH asks for digital proof of your identity. If you pass the test, you may log in; otherwise SSH rejects the connection.

Encryption Scrambles data so it is unintelligible except to the intended recipients. This protects your data as it passes over the network.

Integrity Guarantees the data traveling over the network arrives unaltered. If a third party captures and modifies your data in transit, SSH detects this fact.

The **ssh command uses a ssh protocol**, which is a secure protocol, as the data transfer between the client and the host takes place in encrypted form. It transfers the input through the client to the host and returns the output transferred by the host. It executes through TCP/IP **Port 22.**

The port number can be configured by changing the **Port 22 directive in /etc/ssh/sshd_config**. It can also be specified using the -p <port> option to sshd. **The SSH client and sftp programs also support the -p <port> option.**

3. How SSH Evolved

Few years ago, message or data transfer was not possible and said to give unreliable results. But with the advent of time, day by day technology increases with the increase of human demand. Previously, this message transfer was not possible and is not able to provide correct results. But with the time corrections are made to make it a reliable way of communication with the devices. One of the great researchers **Tatu Ylönen** of **Helsinki University of Finland** planned the initial version of protocol which is called **SSH:1 in 1995.** The main goal of **SSH was to replace the untimely rlogin, TELNET,FTP and rsh protocols**, which did not deliver robust validation nor assurance confidentiality. Ylönen released his implementation as Freeware in July 1995. Ylönen founded the SSH communication Security to market and he developed SSH in December 1995. The main version of the SSH communication uses different parts of software like GNU libgmp, and later varieties released by the SSH progressed into increased software. There are various categories or the versions used in SSH communications: Version 1.x Version 2.x Version 1.99 OpenSSH

In 1998, SCS released the software product "SSH Secure Shell" (SSH2), based on the superior SSH-2 protocol. However, **SSH2 didn't replace SSH1** in the field, for two reasons. First, SSH2 was missing a number of useful, practical features and configuration options of SSH1. Second, SSH2 had a more restrictive license. The original SSH1 had been freely available from Ylönen and the Helsinki University of Technology. **Newer versions of SSH1 from SCS were still freely available for most uses**, even in commercial settings, as long as the software was not directly sold for profit or offered as a service to customers. SSH2, on the other hand, was a commercial product, allowing gratis use only for qualifying educational and non-profit entities. As a result, when SSH2 first appeared, most existing SSH1 users saw few advantages to **SSH2 and continued to use SSH1**. As of this writing, three years after the introduction of the **SSH-2 protocol, SSH-1 is still the most widely deployed version on the Internet, even though SSH-2 is a better and more secure protocol**

4. Why SSH

SSH enables us to provide a service with encrypted access for the widest range of operating systems **(Windows XP-10, Mac OS X and Linux);** this would not be possible if we provided Windows networked drives (which utilize the SMB/CIFS communication protocol). SSH is **reliable and secure** and is often used in the High Performance Computing community for this reason.

The **business environment** is transforming. Enterprises have embarked into a digital transformation journey adopting emerging technologies that allow them to **move fast and change how they collaborate**, reducing costs and increasing productivity. However, these technologies have vanished the traditional perimeter and identity has become the new line of defense. **Modern challenges require modern security approaches**. The use of passwords to authenticate privileged access to mission-critical assets is no longer acceptable. **Passwords are infamous for being insecure**, creating fatigue and a false sense of security. Enterprises need to adopt passwordless solutions – th**is is where the SSH key-based authentication comes in handy.**

People use **SSH to communicate securely with another computer**. By using SSH, the exchange of **data is encrypted across the Internet** pathways. This way anyone who happened to see the data streaming by, would not be able to see what was in the data. The closest alternative to SSH is Telnet. It also lets a user communicate with another computer. But unlike SSH, the exchange of data via Telnet is sent in the clear. So anyone watching the data stream will be able to see all of the **data you send between the two systems**. There are various ways to watch data as it streams across the Internet. The reason for this has to do with the **fundamental protocols that allow any two computers** to find each other within the vast array of systems that make up the Internet. Those protocols involve **one computer asking another if they are the correct recipient for a piece of data or not.** Any computer that receives such a request has access to the data in question. And those requests fly around all the time, everywhere on the Internet

5. Applications of SSH

1. **Symmetrical Encryption** It is a structure of encryption where a classified key is used for encryption and decryption of a data communication by the client and the server. Practically anyone controlling the key can decrypt the message being transferred. In symmetrical encryption one or more keys are used to pass the data from one system to another and moreover the client and the server expand the keys using an agreed method.

2. **Asymmetric Encryption** Apart from other encryption this method basically uses two separate keys for encryption and decryption and these keys are also called public and private keys. Public key is openly distributed and shared in all the devices or systems and more closely connected with the private key with respect to its functionality whereas private key is not shared with the device and its connection is secured and robust in the system.

3 Hashing Hashing is a part and parcel form of cryptography which is basically implemented in the secure shell connectivity[2]. There is no need for private and public keys in this part of hashing. There is one way hash functions which are different from other forms of encryption as they are not suitable for the decryption part of the message from one system to another. Hashing mainly builds a unique value of affixed length in each input that shows the no clear trend that can be destroyed. Moreover it is not possible to build or generate the input from the hash function as the client holds the correct input and checks whether they are accessing the correct value or not[3]. Secure shell access hash function to check the authentication of the data or the message and it is basically done by the help of HMAC's.

FUNCTIONS:-

- It mainly secures remote access towards SSH that enables the network systems and for the users as well as for the automated process.

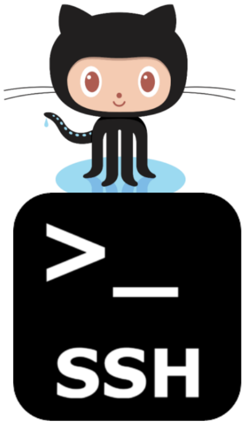
- It also provides interactive file or data transfer sessions. It mainly focuses on issuance of commands on the remote systems.

- It is also secure in management of the network system components.

6. GitHub Using SSH

Connecting to GitHub with SSH

You can connect to GitHub using the Secure Shell Protocol (SSH), which provides a secure channel over an unsecured network.



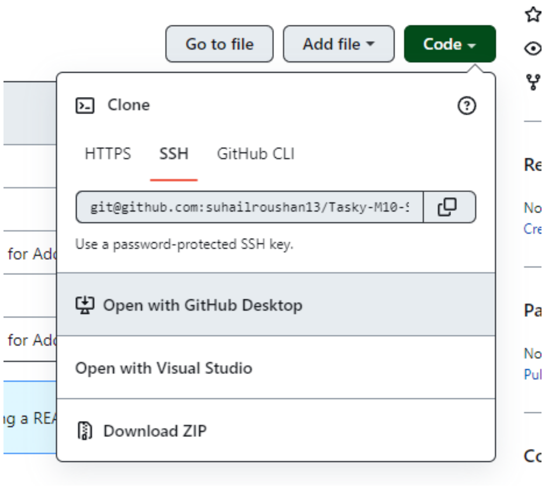
Using the SSH protocol, you can connect and authenticate to remote servers and services. With SSH keys, you can connect to **GitHub without supplying your username and personal access token** at each visit. You can also use an SSH key to **sign commits.**

You can **access and write data in repositories** on GitHub.com using SSH (Secure Shell Protocol). When you connect via SSH, you authenticate using a private key file on your local machine.

You can also secure your SSH key by adding your key to the ssh-agent and using a passphrase. For more information, see "Working with SSH key passphrases." To use your SSH key with a repository owned by an organization that uses SAML single sign-on, you must authorize the key.

Why Use an SSH Key?

When working with a **GitHub repository**, you'll often need to identify yourself to GitHub using your username and password. An SSH key is an alternate way to identify yourself that doesn't require you to enter your username and password every time. S**SH keys come in pairs, a public key that gets shared with services like GitHub, and a private key that is stored only on your computer.** If the keys match, you're granted access. **The cryptography behind SSH keys ensures that no one can reverse engineer your private key from the public one**.



git@github.com:suhailroushan13/Tasky-M10-!

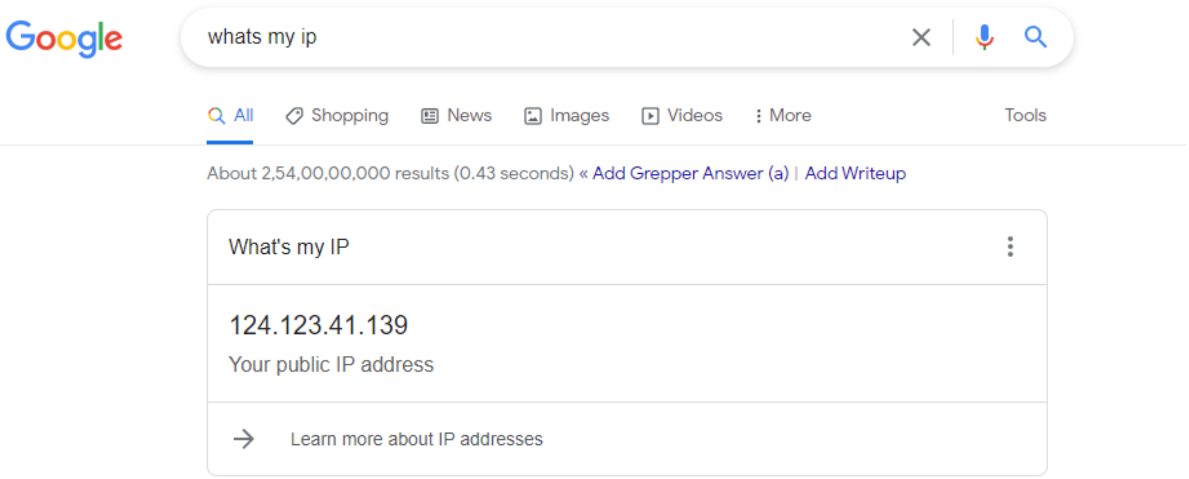
Use a password-protected SSH key.

You can further **secure your SSH key by using a hardware security key**, which requires the physical hardware security key to be attached to your computer when the **key pair is used to authenticate with SSH**.

To maintain account security, you can regularly review your SSH keys list and revoke any keys that are invalid or have been compromised. For more information, see "Reviewing your SSH keys." If you haven't used your SSH key for a year, then **GitHub will automatically delete your inactive SSH key as a security precaution.**

7. Static IP Address

If your computer is hosting a web server, its IP address is what **identifies it to the rest of the Internet.** A computer on the Internet can have a static IP address, which means **it stays the same over time**, or a dynamic IP address, which means the address can change over time.



whats my ip

About 2,54,00,00,000 results (0.43 seconds) « Add Grepper Answer (a) | Add Writeup

What's my IP

124.123.41.139

Your public IP address

Learn more about IP addresses

A s**tatic IP address is a 32 bit number** assigned to a computer as an address on the internet. This number is in the **form of a dotted quad** and is typically **provided by an internet service provider (ISP).** An IP address (internet protocol address) acts as a unique identifier for a device that connects to the internet. **Computers use IP addresses to locate and talk to each other on the internet,** much the same way people use phone numbers to locate and talk to one another on the telephone. **An IP address can provide information such as the hosting provider and geographic location data.**

Note : To Use Private Git Server You Need a Static IP Address

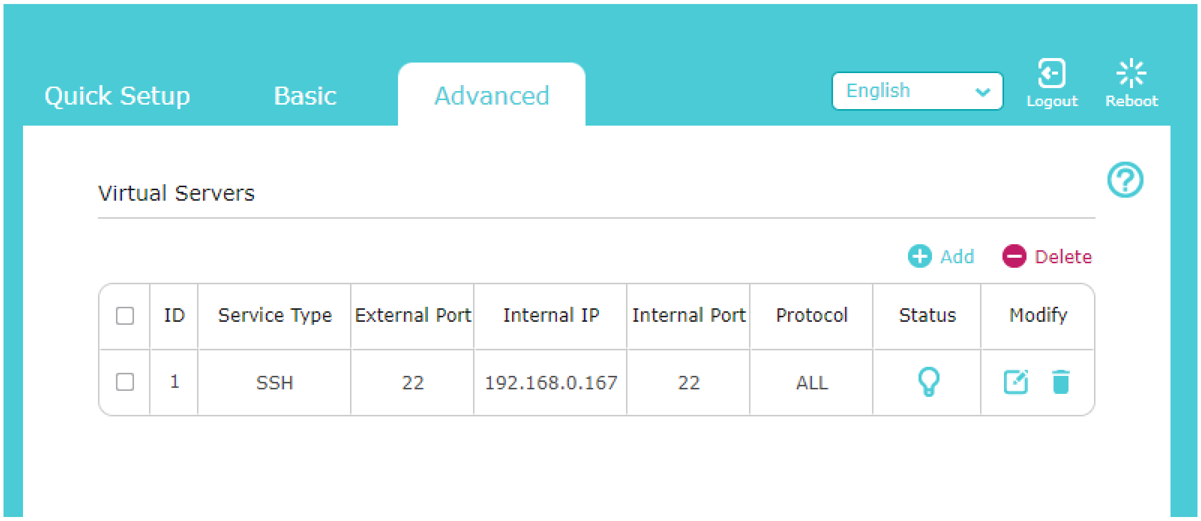
8. Router Table Configurations

Port Forwarding

Port Forwarding **instructs the router to send the request** to a specific PC on your network that will fulfill the request. For example, you tell the router that the PC with an IP address of **192.168.1.103 and an open port:80** is **accepting outside requests**. Setting up **Port Forwarding on your router allows PCs outside the network to access specific services provided by a PC on your network.**

Steps to Allow Port Forwarding

1. Open Your **Router Gateway** [192.168.0.1/1.1]()
2. Click Advanced on the top, then on the left side, click **NAT Forwarding->Virtual Servers->Add.**
3. Add Your Local IP Of Server Machine
   Example **IP of Server Machine** Was [191.168.0.167]()
   And SSH Uses Port Number 22



Virtual Servers

□	ID	Service Type	External Port	Internal IP	Internal Port	Protocol	Status	Modify
□	1	SSH	22	192.168.0.167	22	ALL		

9. Installing OpenSSH Server

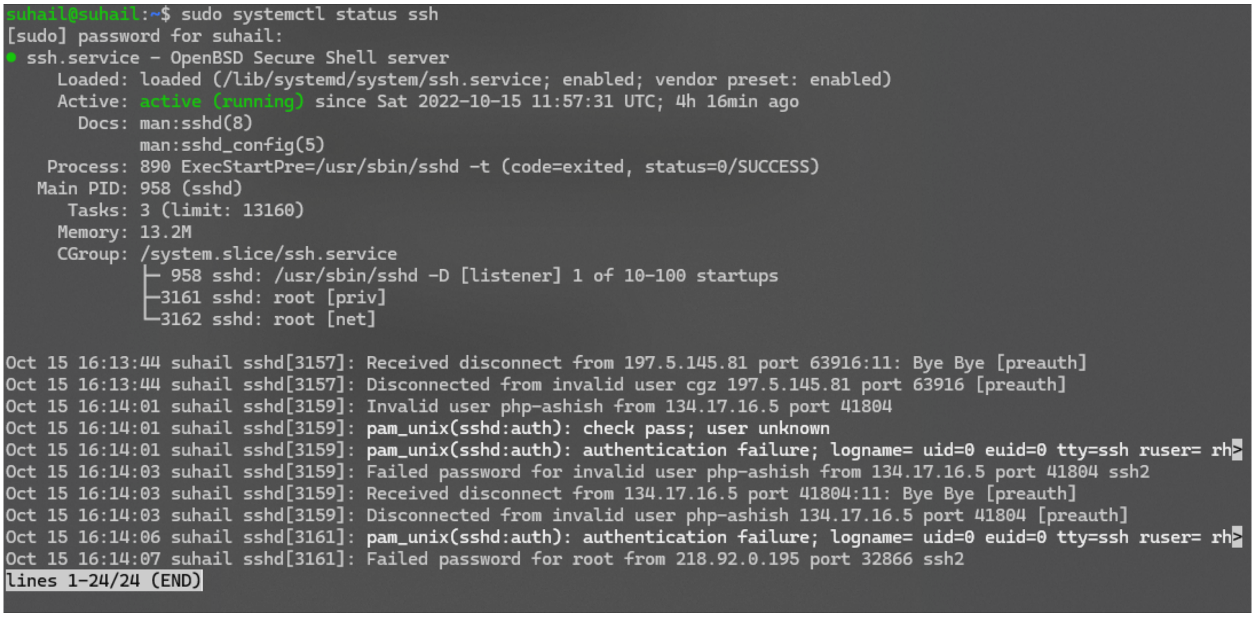
In Windows

```
Add-WindowsCapability -Online -Name OpenSSH.Server
```

In Ubuntu

1. Open the Terminal
2. Type `sudo apt-get install openssh-server`
3. Type `sudo systemctl enable ssh --now`
4. Type `sudo systemctl start ssh`
5. Verify that ssh service running

   Type the following systemctl command: `sudo systemctl status ssh`

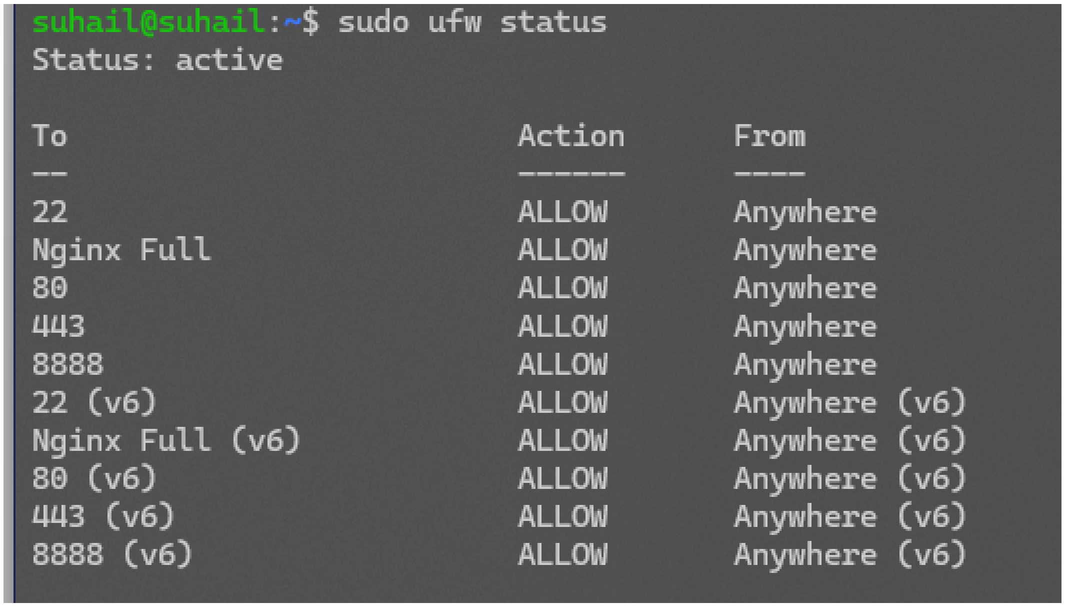


```
suhail@suhail:~$ sudo systemctl status ssh
[sudo] password for suhail:
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2022-10-15 11:57:31 UTC; 4h 16min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 890 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 958 (sshd)
      Tasks: 3 (limit: 13160)
     Memory: 13.2M
     CGroup: /system.slice/ssh.service
             ├─ 958 sshd: /usr/sbin/sshd -D [listener] 1 of 10-100 startups
             ├─3161 sshd: root [priv]
             └─3162 sshd: root [net]

Oct 15 16:13:44 suhail sshd[3157]: Received disconnect from 197.5.145.81 port 63916:11: Bye Bye [preauth]
Oct 15 16:13:44 suhail sshd[3157]: Disconnected from invalid user cgz 197.5.145.81 port 63916 [preauth]
Oct 15 16:14:01 suhail sshd[3159]: Invalid user php-ashish from 134.17.16.5 port 41804
Oct 15 16:14:01 suhail sshd[3159]: pam_unix(sshd:auth): check pass; user unknown
Oct 15 16:14:01 suhail sshd[3159]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh>
Oct 15 16:14:03 suhail sshd[3159]: Failed password for invalid user php-ashish from 134.17.16.5 port 41804 ssh2
Oct 15 16:14:03 suhail sshd[3159]: Received disconnect from 134.17.16.5 port 41804:11: Bye Bye [preauth]
Oct 15 16:14:03 suhail sshd[3159]: Disconnected from invalid user php-ashish 134.17.16.5 port 41804 [preauth]
Oct 15 16:14:06 suhail sshd[3161]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh>
Oct 15 16:14:07 suhail sshd[3161]: Failed password for root from 218.92.0.195 port 32866 ssh2
lines 1-24/24 (END)
```

10. Enabling Firewall Port

Configure firewall and open port 22 In Ubuntu

```
sudo ufw allow ssh
sudo ufw enable
sudo ufw status
```



```
suhail@suhail:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
22                         ALLOW       Anywhere
Nginx Full                 ALLOW       Anywhere
80                         ALLOW       Anywhere
443                        ALLOW       Anywhere
8888                       ALLOW       Anywhere
22 (v6)                    ALLOW       Anywhere (v6)
Nginx Full (v6)            ALLOW       Anywhere (v6)
80 (v6)                    ALLOW       Anywhere (v6)
443 (v6)                   ALLOW       Anywhere (v6)
8888 (v6)                  ALLOW       Anywhere (v6)
```

Make Sure Port 22 is Updated

11. SSH Keys

Generating SSH Keys Using Algorithms

Configures **SSH to use a set of host key algorithms** in the specified priority order. Host key algorithms specify which host key types are allowed to be used for the SSH connection. The first host key entered in the CLI is considered a first priority. Each option represents a type of key that can be used. Host keys are used to verify the host that you are connecting to. This configuration allows you to control which host key types are presented to incoming clients, or which host key types to receive first from hosts. Only the host key algorithms that are specified by the user are configured.

The no form of this command removes the configuration of host key algorithms and reverts SSH to use the default set of algorithms.

Types of Algorithms

Default set of host key algorithms in priority order:

1. ecdsa-sha2-nistp256
2. ecdsa-sha2-nistp384
3. ecdsa-sha2-nistp521
4. ssh-ed25519
5. rsa-sha2-256
6. rsa-sha2-512
7. ssh-rsa

Choosing an Algorithm and Key Size

SSH supports several public key algorithms for authentication keys. These include:

- **rsa** - an old algorithm based on the difficulty of factoring large numbers. A key size of at least 2048 bits is recommended for RSA; 4096 bits is better. RSA is getting old and significant advances are being made in factoring. Choosing a different algorithm may be advisable. It is quite possible the RSA algorithm will become practically breakable in the foreseeable future. All SSH clients support this algorithm.
- **dsa -** an old US government Digital Signature Algorithm. It is based on the difficulty of computing discrete logarithms. A key size of 1024 would normally be used with it. DSA in its original form is no longer recommended.
- **ecdsa** - a new Digital Signature Algorithm standardized by the US government, using elliptic curves. This is probably a good algorithm for current applications. Only three key sizes are supported: 256, 384, and 521 (sic!) bits. We would recommend always using it with 521 bits, since the keys are still small and probably more secure than the smaller keys (even though they should be safe as well). Most SSH clients now support this algorithm.
- **ed25519** - this is a new algorithm added in OpenSSH. Support for it in clients is not yet universal. Thus its use in general purpose applications may not yet be advisable.

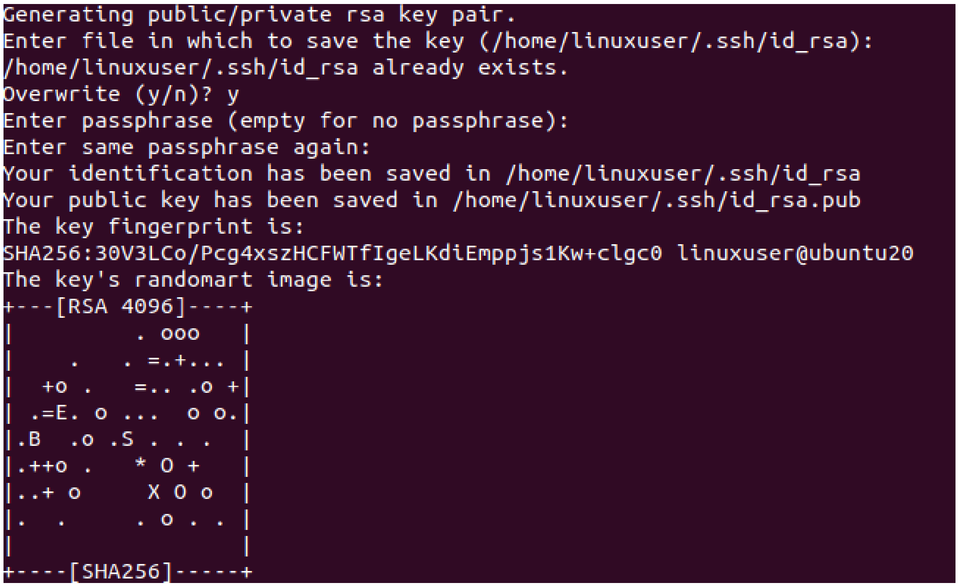
The algorithm is selected using the -t option and key size using the -b option. The following commands illustrate:

```
ssh-keygen -t rsa -b 4096
ssh-keygen -t dsa
ssh-keygen -t ecdsa -b 521
ssh-keygen -t ed25519
```

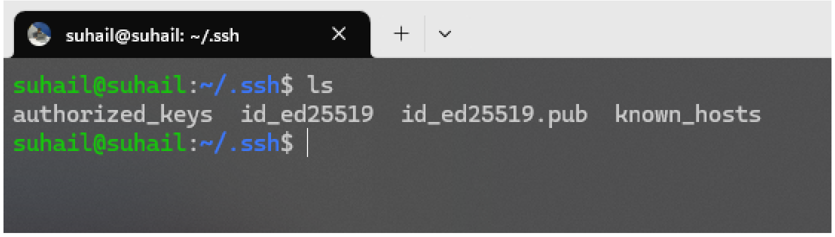
For Now Using Algorithm **ssh-ed25519**

To Generate SSH Keys

1. **ssh-keygen -t ed25519 -C "your_email@example.com"**
2. **If you don't want to set passphrase just hit Enter 3 times**



```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/linuxuser/.ssh/id_rsa):
/home/linuxuser/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/linuxuser/.ssh/id_rsa
Your public key has been saved in /home/linuxuser/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:30V3LCo/Pcg4xszHCFWTfIgeLKdiEmppjs1Kw+clgc0 linuxuser@ubuntu20
The key's randomart image is:
+---[RSA 4096]----+
|         . ooo   |
|    .   . =.+... |
|  +o .   =.. .o +|
| .=E. o ...  o o.|
|.B  .o .S . . .  |
|.++o .   * O +   |
|..+ o     X O o  |
|.  .     . o . . |
|                 |
+----[SHA256]-----+
```

3. **Your SSH Keys has be Generated at /home/{username}/.ssh/**
4. **There are 2 Keys in .ssh Folder**
5. 

```
suhail@suhail:~/.ssh$ ls
authorized_keys  id_ed25519  id_ed25519.pub  known_hosts
suhail@suhail:~/.ssh$
```

6. **id_ed25519.pub is a Public Key**
7. **id_ed25519 is a Private Key**
8. **authrized_keys is file where you can add public keys of Authorized Users**

12. Connect SSH with (Password)

Before Connecting With Set Permissions of Keys

Setting up Permission on SSH Folder and Keys

Type this command’s **Line By Line**

```
sudo chmod 700 .ssh
```

```
 cd .ssh
```

```
 sudo chmod 644 id_ed25519.pub
 sudo chmod 600 id_ed25519
```

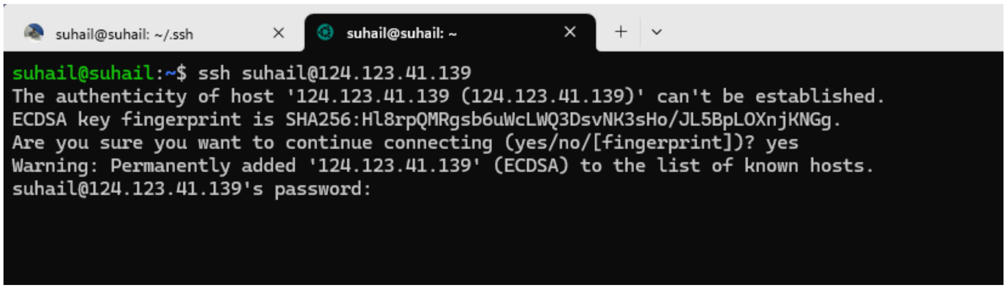
Now Client Connecting Server Using SSH With Password

Connect to your server at its IP address via SSH with the user you would like to add your key to:

ssh server_username@server_static_ip

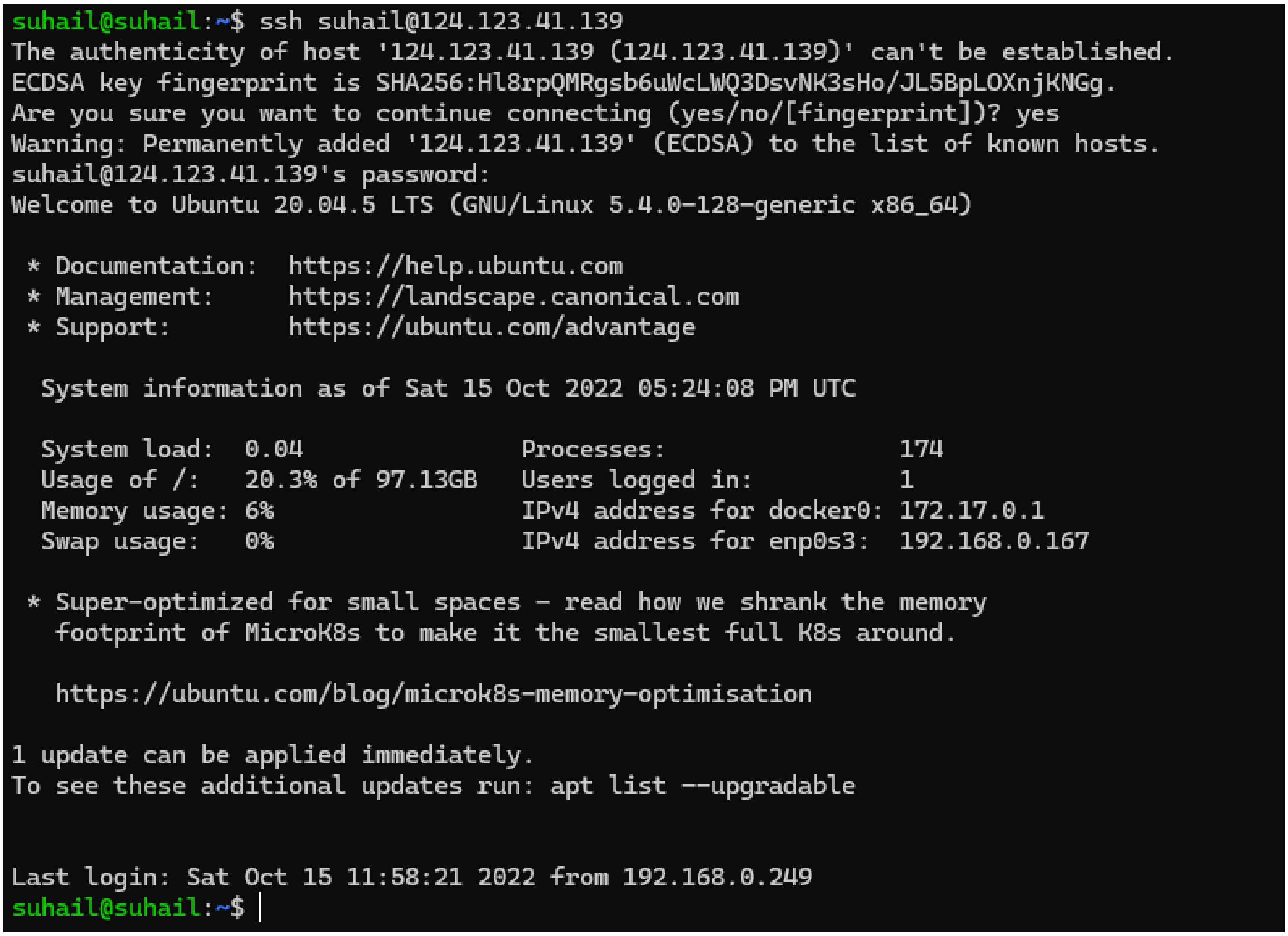
Example :-

ssh suhail@124.123.41.139



```
suhail@suhail:~$ ssh suhail@124.123.41.139
The authenticity of host '124.123.41.139 (124.123.41.139)' can't be established.
ECDSA key fingerprint is SHA256:Hl8rpQMRgsb6uWcLWQ3DsvNK3sHo/JL5BpLOXnjKNGg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '124.123.41.139' (ECDSA) to the list of known hosts.
suhail@124.123.41.139's password:
```

After Entering Password



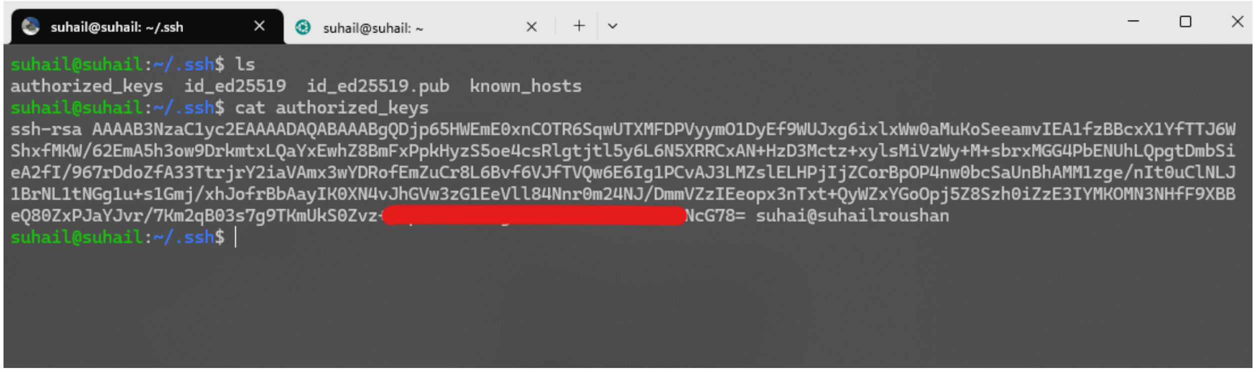
```
suhail@suhail:~$ ssh suhail@124.123.41.139
The authenticity of host '124.123.41.139 (124.123.41.139)' can't be established.
ECDSA key fingerprint is SHA256:Hl8rpQMRgsb6uWcLWQ3DsvNK3sHo/JL5BpLOXnjKNGg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '124.123.41.139' (ECDSA) to the list of known hosts.
suhail@124.123.41.139's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-128-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sat 15 Oct 2022 05:24:08 PM UTC

  System load:  0.04               Processes:                174
  Usage of /:   20.3% of 97.13GB   Users logged in:          1
  Memory usage: 6%                 IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                 IPv4 address for enp0s3:  192.168.0.167

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable


Last login: Sat Oct 15 11:58:21 2022 from 192.168.0.249
suhail@suhail:~$ 
```

We successfully connected to Server Using SSH Password

13.Authorized Keys

An authorized key in SSH is a public key used for granting login access to users. The authentication mechanism is called public key authentication.

Authorized keys are configured separately for each user - usually in the .ssh/authorized_keys file in the user's home directory. However, the location of the keys can be configured in SSH server configuration files, and is often changed to a root-owned location in more secure environments.



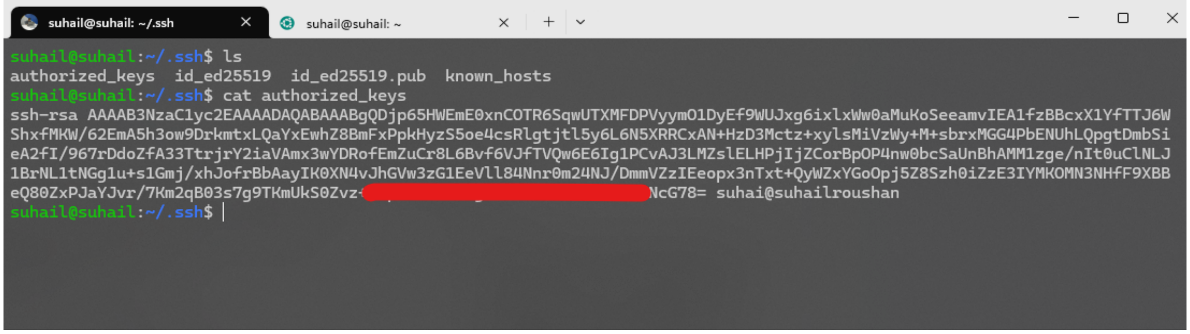
```
suhail@suhail:~/.ssh$ ls
authorized_keys  id_ed25519  id_ed25519.pub  known_hosts
suhail@suhail:~/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDjp65HWEmE0xnCOTR6SqwUTXMFDPVyymO1DyEf9WUJxg6ixlxWw0aMuKoSeeamvIEA1fzBBcxX1YfTTJ6W
ShxfMKW/62EmA5h3ow9DrkmtxLQaYxEwhZ8BmFxPpkHyzS5oe4csRlgtjtl5y6L6N5XRRCxAN+HzD3Mctz+xylsMiVzWy+M+sbrxMGG4PbENUhLQpgtDmbSi
eA2fI/967rDdoZfA33TtrjrY2iaVAmx3wYDRofEmZuCr8L6Bvf6VJfTVQw6E6Ig1PCvAJ3LMZslELHPjIjZCorBpOP4nw0bcSaUnBhAMM1zge/nIt0uClNLJ
1BrNL1tNGg1u+s1Gmj/xhJofrBbAayIK0XN4vJhGVw3zG1EeVll84Nnr0m24NJ/DmmVZzIEeopx3nTxt+QyWZxYGoOpj5Z8Szh0iZzE3IYMKOMN3NHfF9XBB
eQ80ZxPJaYJvr/7Km2qB03s7g9TKmUkS0Zvz[redacted]NcG78= suhai@suhailroushan
suhail@suhail:~/.ssh$ 
```

14.Connect SSH with Public Key (Passwordless)

I will share my public key with you
Ok then i will share my terminal access to you
KAPWING

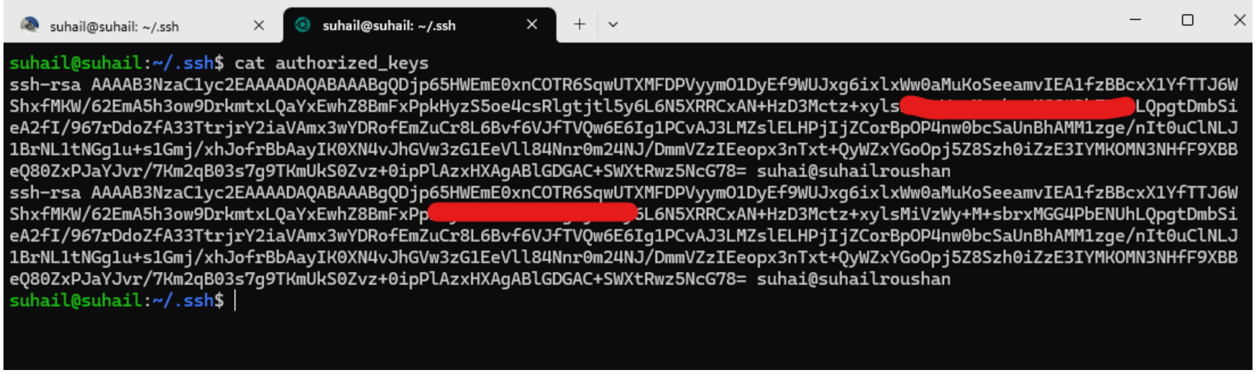
Client Sharing Public Key With Server Gives The Terminal Access to Client

1.Get your Client Public key which is **id_ed25519.pub**
2.Copy the key from file



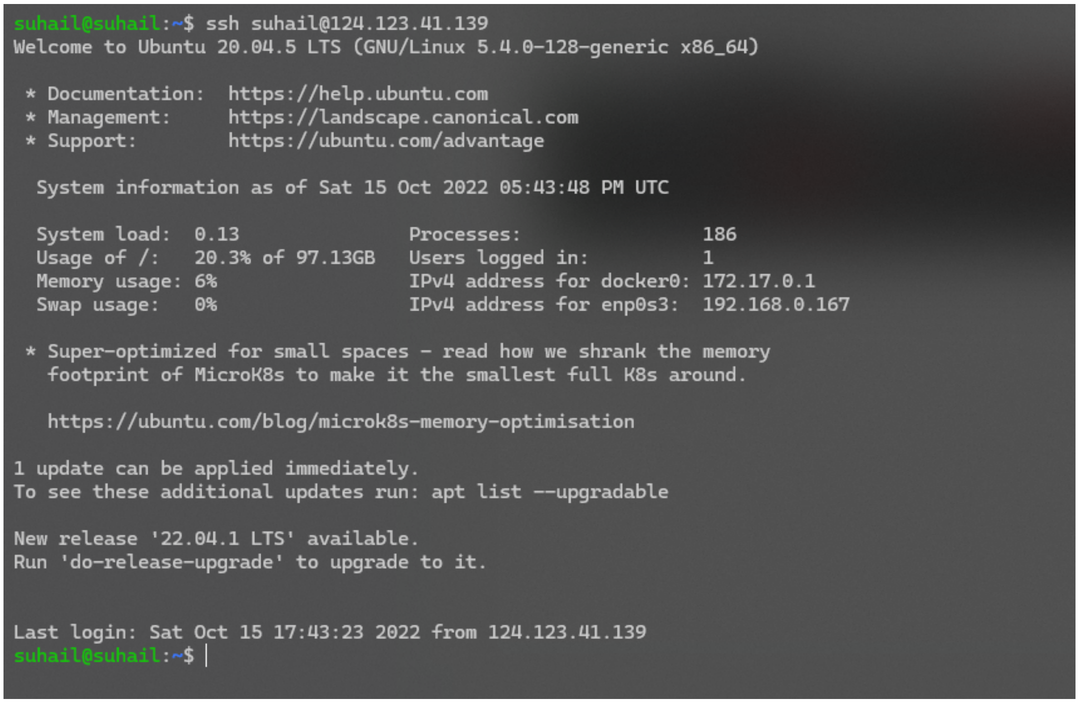
```
suhail@suhail:~/.ssh$ ls
authorized_keys  id_ed25519  id_ed25519.pub  known_hosts
suhail@suhail:~/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDjp65HWEmE0xnCOTR6SqwUTXMFDPVyymO1DyEf9WUJxg6ixlxWw0aMuKoSeeamvIEA1fzBBcxX1YfTTJ6W
ShxfMKW/62EmA5h3ow9DrkmtxLQaYxEwhZ8BmFxPpkHyzS5oe4csRlgtjtl5y6L6N5XRRCxAN+HzD3Mctz+xylsMiVzWy+M+sbrxMGG4PbENUhLQpgtDmbSi
eA2fI/967rDdoZfA33TtrjrY2iaVAmx3wYDRofEmZuCr8L6Bvf6VJfTVQw6E6Ig1PCvAJ3LMZslELHPjIjZCorBpOP4nw0bcSaUnBhAMM1zge/nIt0uClNLJ
1BrNL1tNGg1u+s1Gmj/xhJofrBbAayIK0XN4vJhGVw3zG1EeVll84Nnr0m24NJ/DmmVZzIEeopx3nTxt+QyWZxYGoOpj5Z8Szh0iZzE3IYMKOMN3NHfF9XBB
eQ80ZxPJaYJvr/7Km2qB03s7g9TKmUkS0Zvz[redacted]NcG78= suhai@suhailroushan
suhail@suhail:~/.ssh$ |
```

Paste this Public Key In Server Authorized Key File



```
suhail@suhail:~/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDjp65HWEmE0xnCOTR6SqwUTXMFDPVyymO1DyEf9WUJxg6ixlxWw0aMuKoSeeamvIEA1fzBBcxX1YfTTJ6W
ShxfMKW/62EmA5h3ow9DrkmtxLQaYxEwhZ8BmFxPpkHyzS5oe4csRlgtjtl5y6L6N5XRRCxAN+HzD3Mctz+xyls[redacted]LQpgtDmbSi
eA2fI/967rDdoZfA33TtrjrY2iaVAmx3wYDRofEmZuCr8L6Bvf6VJfTVQw6E6Ig1PCvAJ3LMZslELHPjIjZCorBpOP4nw0bcSaUnBhAMM1zge/nIt0uClNLJ
1BrNL1tNGg1u+s1Gmj/xhJofrBbAayIK0XN4vJhGVw3zG1EeVll84Nnr0m24NJ/DmmVZzIEeopx3nTxt+QyWZxYGoOpj5Z8Szh0iZzE3IYMKOMN3NHfF9XBB
eQ80ZxPJaYJvr/7Km2qB03s7g9TKmUkS0Zvz+0ipPlAzxHXAgABlGDGAC+SWXtRwz5NcG78= suhai@suhailroushan
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDjp65HWEmE0xnCOTR6SqwUTXMFDPVyymO1DyEf9WUJxg6ixlxWw0aMuKoSeeamvIEA1fzBBcxX1YfTTJ6W
ShxfMKW/62EmA5h3ow9DrkmtxLQaYxEwhZ8BmFxPp[redacted]6L6N5XRRCxAN+HzD3Mctz+xylsMiVzWy+M+sbrxMGG4PbENUhLQpgtDmbSi
eA2fI/967rDdoZfA33TtrjrY2iaVAmx3wYDRofEmZuCr8L6Bvf6VJfTVQw6E6Ig1PCvAJ3LMZslELHPjIjZCorBpOP4nw0bcSaUnBhAMM1zge/nIt0uClNLJ
1BrNL1tNGg1u+s1Gmj/xhJofrBbAayIK0XN4vJhGVw3zG1EeVll84Nnr0m24NJ/DmmVZzIEeopx3nTxt+QyWZxYGoOpj5Z8Szh0iZzE3IYMKOMN3NHfF9XBB
eQ80ZxPJaYJvr/7Km2qB03s7g9TKmUkS0Zvz+0ipPlAzxHXAgABlGDGAC+SWXtRwz5NcG78= suhai@suhailroushan
suhail@suhail:~/.ssh$ 
```

Now Try to Connect



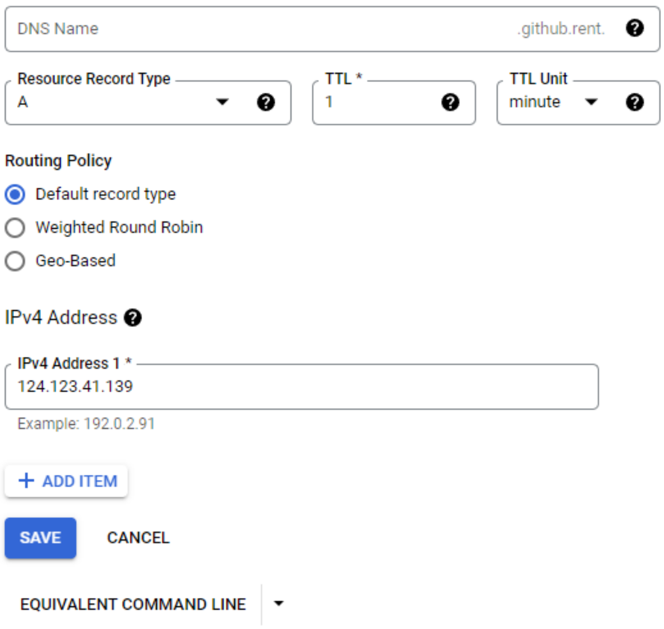
```
suhail@suhail:~$ ssh suhail@124.123.41.139
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-128-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sat 15 Oct 2022 05:43:48 PM UTC

  System load:  0.13              Processes:                186
  Usage of /:   20.3% of 97.13GB  Users logged in:          1
  Memory usage: 6%                IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                IPv4 address for enp0s3:  192.168.0.167

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Sat Oct 15 17:43:23 2022 from 124.123.41.139
suhail@suhail:~$ 
```

It Got Connected Without Asking For Password

15.IP Address to Domain

An **A record maps a domain name to the IP address (Version 4)** of the computer hosting the domain. An A record uses a **domain name to find the IP address of a computer connected to the internet** The **A in A record stands for Address**. Whenever you visit a web site, send an email, connect to Twitter or Facebook, or do almost anything on the Internet, the address you enter is a series of words connected with dots. For example, to access the DNSimple website you enter www.dnsimple.com. At our name server, there's an A record that points to the IP **address 208.93.64.253. This means that a request from your browser to www.dnsimple.com** is directed to the server with IP address 208.93.64.253. A Records are the simplest type of DNS records, and one of the primary records used in DNS servers. You can do a lot with A records, including using multiple A records for the same domain in order to provide redundancy and fallbacks. Additionally, multiple names could point to the same address, in which case each would have its own **A record pointing to that same IP address**.
The DNS A record is specified by RFC 1035.

Static IP Add in A Record In Your Hosting DNS Settings



DNS Name .github.rent.

Resource Record Type: A

TTL *: 1

TTL Unit: minute

Routing Policy

● Default record type

○ Weighted Round Robin

○ Geo-Based

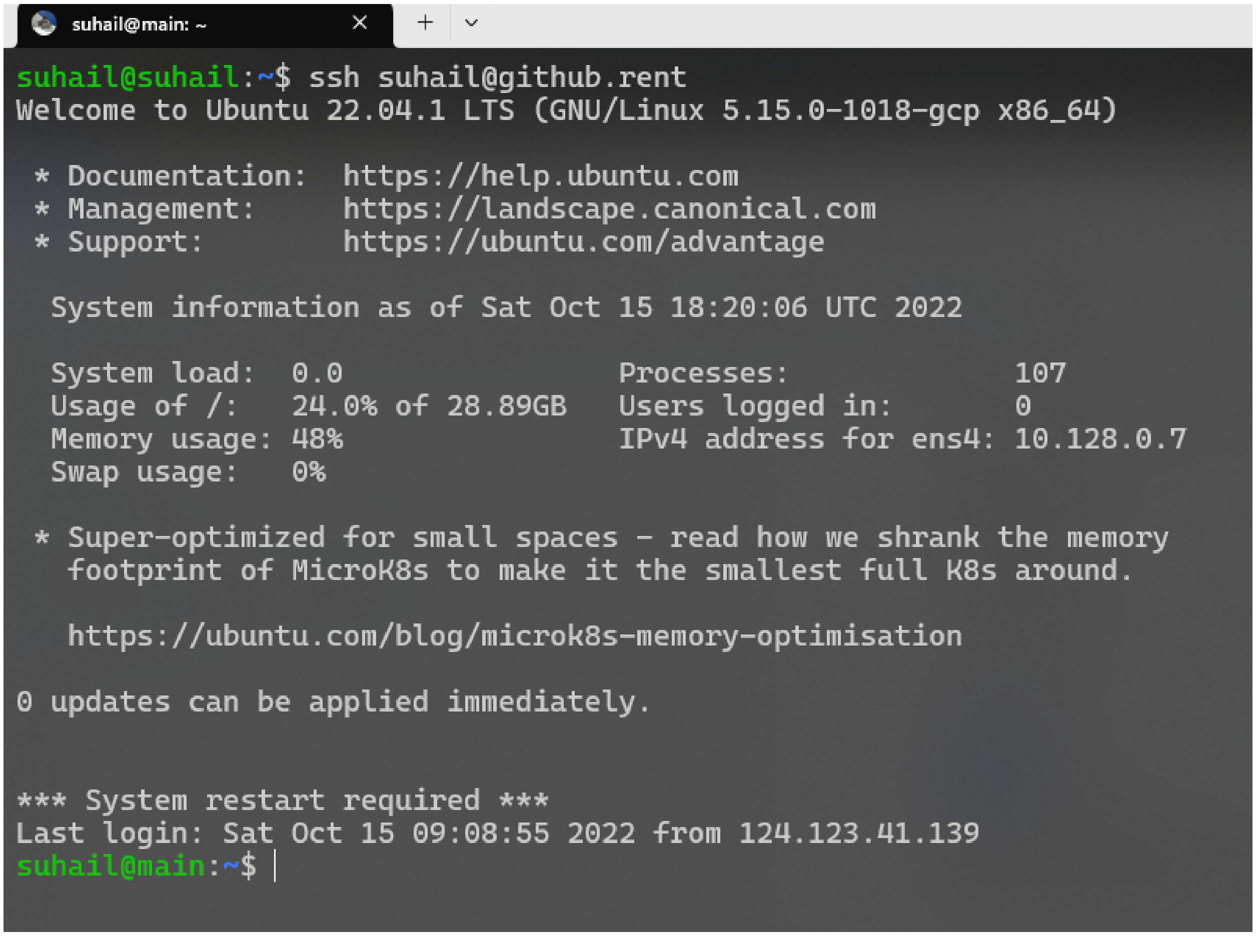
IPv4 Address

IPv4 Address 1 *: 124.123.41.139

Example: 192.0.2.91

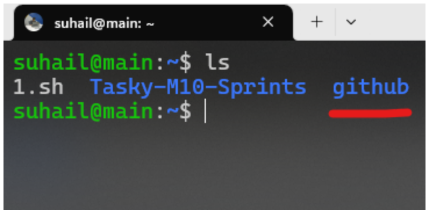
Static IP : 124.123.41.139 ⇒ github.rent

Connecting SSH With Domain



```
suhail@suhail:~$ ssh suhail@github.rent
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-1018-gcp x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sat Oct 15 18:20:06 UTC 2022

  System load:  0.0                Processes:             107
  Usage of /:   24.0% of 28.89GB   Users logged in:       0
  Memory usage: 48%                IPv4 address for ens4: 10.128.0.7
  Swap usage:   0%

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

0 updates can be applied immediately.


*** System restart required ***
Last login: Sat Oct 15 09:08:55 2022 from 124.123.41.139
suhail@main:~$ 
```

16. Git Server Initialization

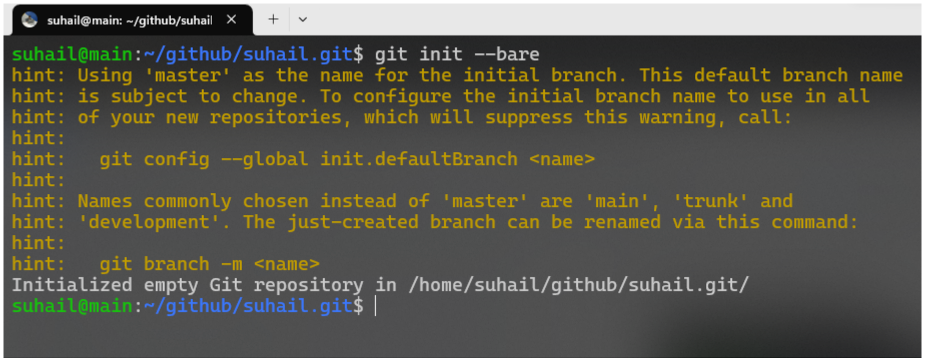
1. Open Your Server Machine
2. Create a folder a github
3. 

```
suhail@main:~$ ls
1.sh  Tasky-M10-Sprints  github
suhail@main:~$ |
```

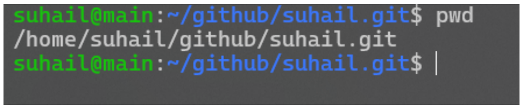
4. cd github
5. Create github usernames
6. Example :- mkdir suhail.git ishaan.git sara.git adnan.git
7. Creating .git folder makes available for the users for there repos
8. cd suhail.git

Bare Git Repo The other variant creates a repository without a working directory (git init --bare). You don't get a directory where you can work. Everything in the directory is now what was contained in the .git folder in the above case.

Bare Git Repo acts like you open your server to act like Hosting Where Client can access there Repos

9. git init –bare
10. 

```
suhail@main:~/github/suhail.git$ git init --bare
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/suhail/github/suhail.git/
suhail@main:~/github/suhail.git$ |
```

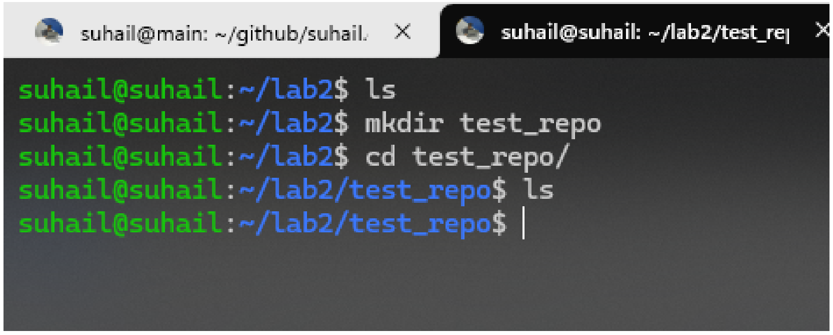
11. Copy The Path of username.git
12. 

```
suhail@main:~/github/suhail.git$ pwd
/home/suhail/github/suhail.git
suhail@main:~/github/suhail.git$ |
```

13. **Path is :** /home/suhail/github/suhail.git

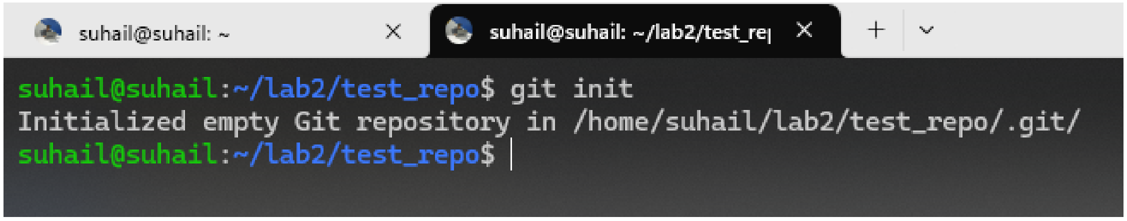
17. Sending Files In Git Server

1. Open Your Client Machine
2. Create a empty folder name it



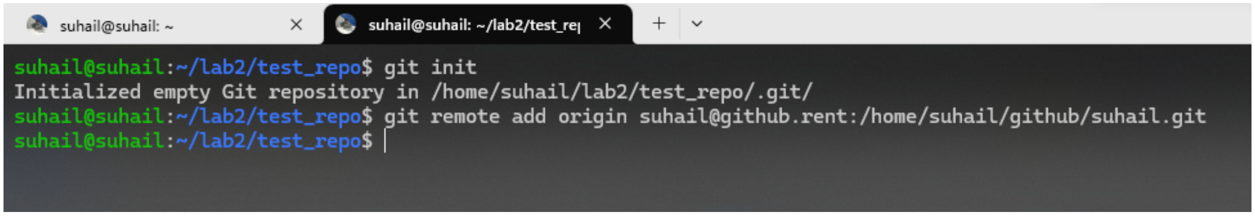
```
suhail@suhail:~/lab2$ ls
suhail@suhail:~/lab2$ mkdir test_repo
suhail@suhail:~/lab2$ cd test_repo/
suhail@suhail:~/lab2/test_repo$ ls
suhail@suhail:~/lab2/test_repo$ |
```

3.
4. Initialize the repo git init



```
suhail@suhail:~/lab2/test_repo$ git init
Initialized empty Git repository in /home/suhail/lab2/test_repo/.git/
suhail@suhail:~/lab2/test_repo$ |
```

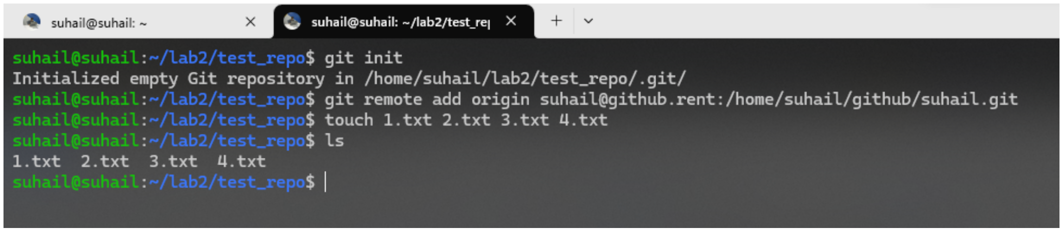
5.
6. Add Remote
7. git remote add origin suhail@github.rent:/home/suhail/github/suhail.git



```
suhail@suhail:~/lab2/test_repo$ git init
Initialized empty Git repository in /home/suhail/lab2/test_repo/.git/
suhail@suhail:~/lab2/test_repo$ git remote add origin suhail@github.rent:/home/suhail/github/suhail.git
suhail@suhail:~/lab2/test_repo$ |
```

8.
9. Where
   suhail is the hostname
   github.rent is the host ip address attached with domain using A Record
   /home/suhail/github/suhail.git is the path of the user folder

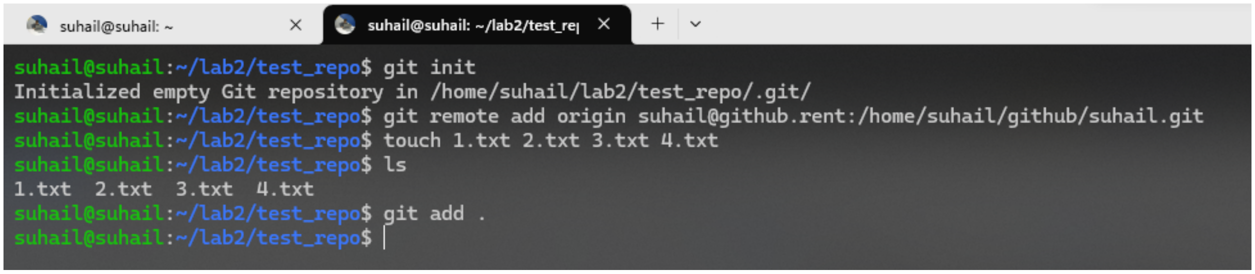
10. Create files to be pushed in server



```
suhail@suhail:~/lab2/test_repo$ git init
Initialized empty Git repository in /home/suhail/lab2/test_repo/.git/
suhail@suhail:~/lab2/test_repo$ git remote add origin suhail@github.rent:/home/suhail/github/suhail.git
suhail@suhail:~/lab2/test_repo$ touch 1.txt 2.txt 3.txt 4.txt
suhail@suhail:~/lab2/test_repo$ ls
1.txt  2.txt  3.txt  4.txt
suhail@suhail:~/lab2/test_repo$ |
```

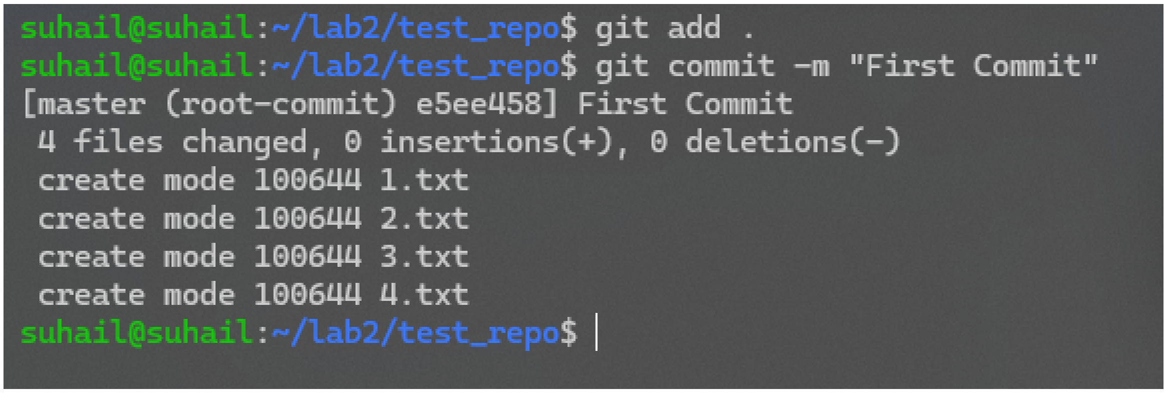
11. git add.

The git add command adds a change in the working directory to the staging area.



```
suhail@suhail:~/lab2/test_repo$ git init
Initialized empty Git repository in /home/suhail/lab2/test_repo/.git/
suhail@suhail:~/lab2/test_repo$ git remote add origin suhail@github.rent:/home/suhail/github/suhail.git
suhail@suhail:~/lab2/test_repo$ touch 1.txt 2.txt 3.txt 4.txt
suhail@suhail:~/lab2/test_repo$ ls
1.txt  2.txt  3.txt  4.txt
suhail@suhail:~/lab2/test_repo$ git add .
suhail@suhail:~/lab2/test_repo$
```

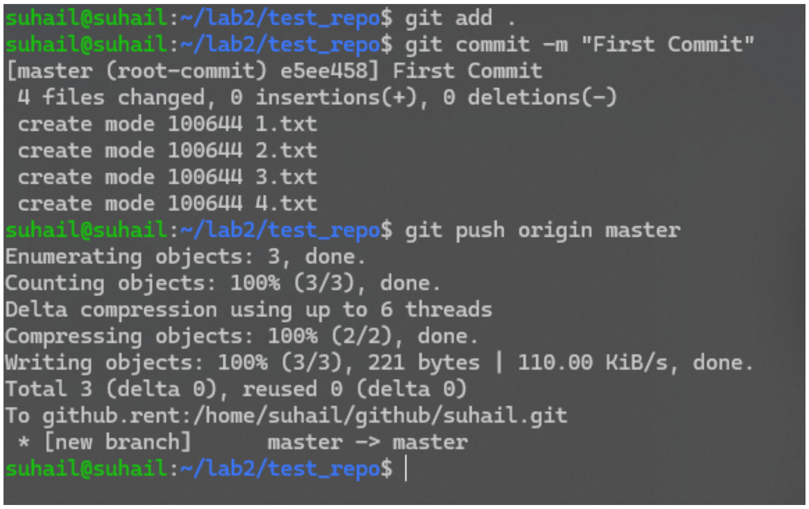
12. git commit -m “First Commit” The git commit command captures a snapshot of the project's currently staged changes



```
suhail@suhail:~/lab2/test_repo$ git add .
suhail@suhail:~/lab2/test_repo$ git commit -m "First Commit"
[master (root-commit) e5ee458] First Commit
 4 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1.txt
 create mode 100644 2.txt
 create mode 100644 3.txt
 create mode 100644 4.txt
suhail@suhail:~/lab2/test_repo$
```

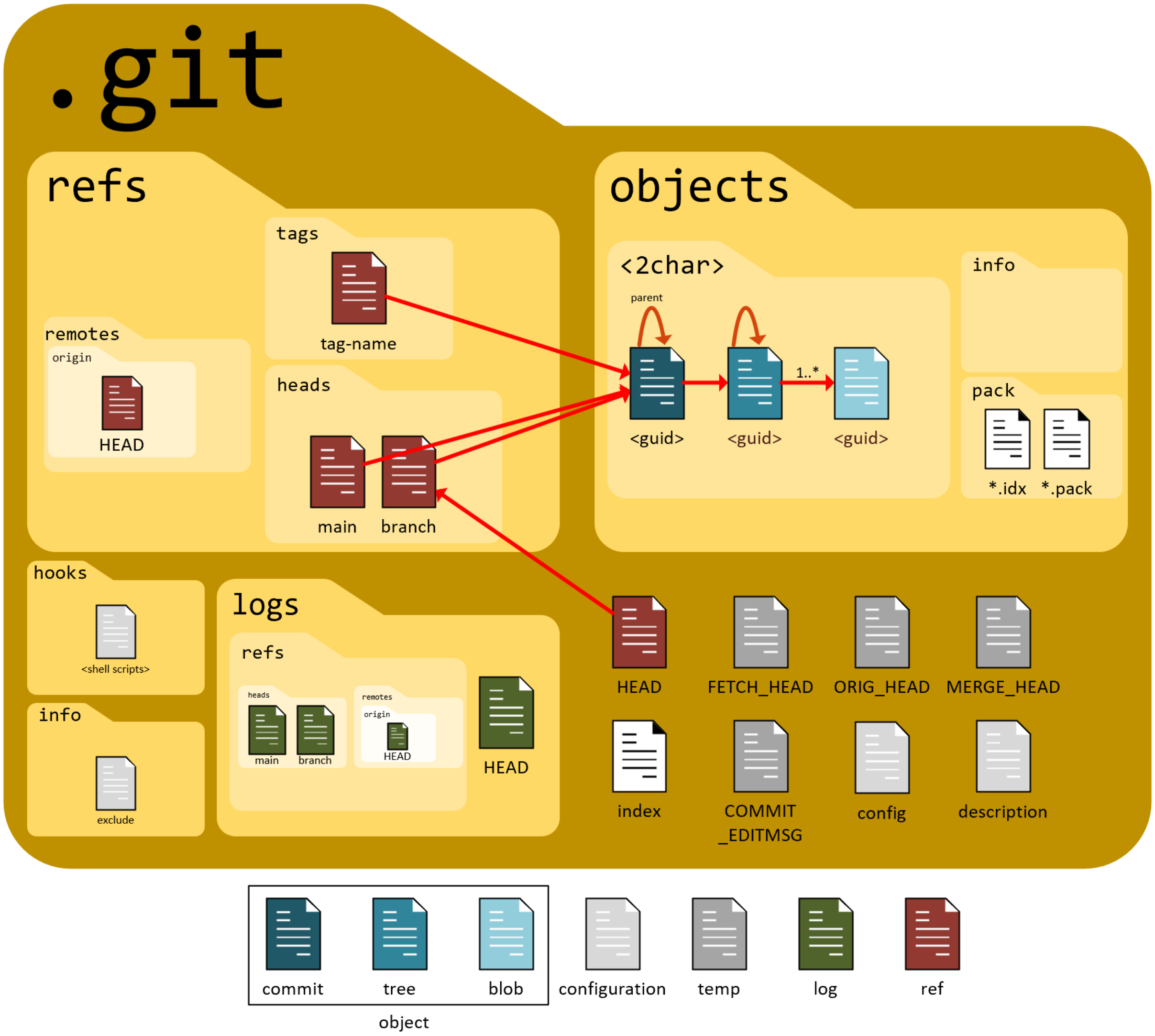
13. git push origin master

The git push command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo.



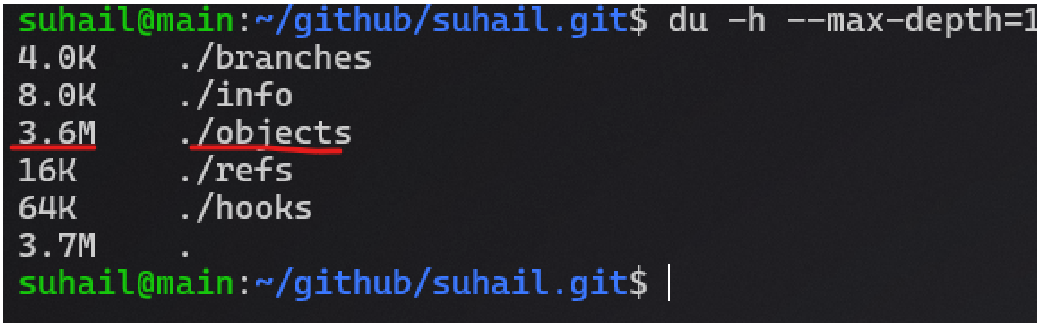
```
suhail@suhail:~/lab2/test_repo$ git add .
suhail@suhail:~/lab2/test_repo$ git commit -m "First Commit"
[master (root-commit) e5ee458] First Commit
 4 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1.txt
 create mode 100644 2.txt
 create mode 100644 3.txt
 create mode 100644 4.txt
suhail@suhail:~/lab2/test_repo$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 6 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 221 bytes | 110.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.rent:/home/suhail/github/suhail.git
 * [new branch]      master -> master
suhail@suhail:~/lab2/test_repo$
```

18. Git Objects (Database for Git)

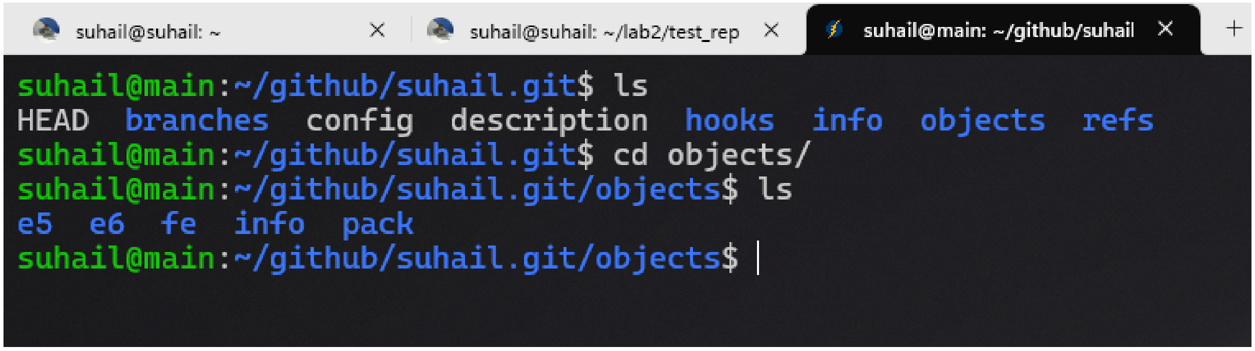


Git is a content-addressable filesystem.
It means that at the core of Git is a **simple key-value data store.**
This means that **you can insert any kind of content** into a Git repository, for which **Git will hand you back a unique key you can use later to retrieve that content.** As a demonstration, let's look at the plumbing command git hash-object, which takes some data, **stores it in your .git/objects directory (the object database),** and gives you back the unique key that now refers to that data object. First, you initialize a new **Git repository and verify that there is (predictably) nothing in the objects directory**

19.Hashed Database in Git



```
suhail@main:~/github/suhail.git$ du -h --max-depth=1
4.0K    ./branches
8.0K    ./info
3.6M    ./objects
16K     ./refs
64K     ./hooks
3.7M    .
suhail@main:~/github/suhail.git$ |
```



```
suhail@main:~/github/suhail.git$ ls
HEAD  branches  config  description  hooks  info  objects  refs
suhail@main:~/github/suhail.git$ cd objects/
suhail@main:~/github/suhail.git/objects$ ls
e5  e6  fe  info  pack
suhail@main:~/github/suhail.git/objects$ |
```

If you again examine your objects directory, you can see that **it now contains a file for that new content**. This is how Git stores the content initially — as a single file per piece of content, named with the SHA-1 checksum of the content and its header. **The subdirectory is named with the first 2 characters of the SHA-1, and the filename is the remaining 38 characters.** Once you have content in your object database, you can examine that content with the git cat-file command. **This command is sort of a Swiss army knife for inspecting Git objects. Passing -p to cat-file instructs the command to first figure out the type of content, then display it appropriately**

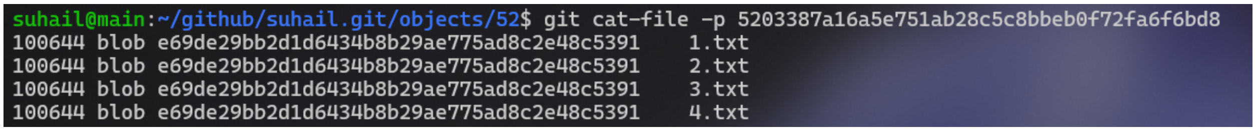
e5 e6 fe are the Hashed folder’s

In its simplest form, git hash-object would take the content you handed to it and merely return the unique key that would be used to store it in your Git database. The -w option then tells the command to not simply return the key, but to write that object to the database. the command would expect a filename argument at the end of the command containing the content to be used. The output from the above command is a 40-character checksum hash. This is the SHA-1 hash — a checksum of the content you’re storing plus a header, which you’ll learn about in a bit. Now you can see how Git has stored your data:

20. Files From Client

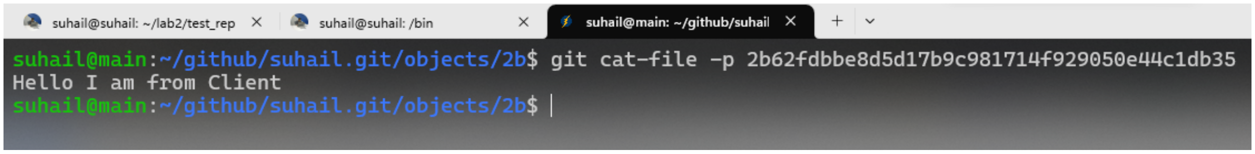
If you again examine your objects directory, you can see that it now contains a file for that new content. This is how Git stores the content initially — as a single file per piece of content, named with the SHA-1 checksum of the content and its header. The subdirectory is named with the first 2 characters of the SHA-1, and the filename is the remaining 38 characters. Once you have content in your object database, you can examine that content with the git cat-file command. This command is sort of a Swiss army knife for inspecting Git objects. Passing -p to cat-file instructs the command to first figure out the type of content, then display it appropriately

But remembering the SHA-1 key for each version of your file isn't practical; plus, you aren't storing the filename in your system — just the content. This object type is called a blob. You can have Git tell you the object type of any object in Git, given its SHA-1 key, with git cat-file -t:



```
suhail@main:~/github/suhail.git/objects/52$ git cat-file -p 5203387a16a5e751ab28c5c8bbeb0f72fa6f6bd8
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    1.txt
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    2.txt
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    3.txt
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    4.txt
```

To See the Content inside the file



```
suhail@main:~/github/suhail.git/objects/2b$ git cat-file -p 2b62fdbbe8d5d17b9c981714f929050e44c1db35
Hello I am from Client
suhail@main:~/github/suhail.git/objects/2b$ |
```

THE-END