# SSH – Secure Shell

**Reference:** <https://help.ubuntu.com/community/SSH/OpenSSH/Configuring>

The Secure Shell protocol (SSH) is used for logging into remote machines, and for transferring files to other machines. There are other protocols for these – telnet is used for logging into remote machines, and rcp, scp, or ftp are used for copying files.

The problem with these other protocols (tenet, etc) is that they are insecure. Files (or commands, passwords, etc) are transferred over the network in plain text, and anyone on the network can see what’s being transferred. Bad idea!

SSH uses encryption to ensure that all communication over the network is secure.

Linux typically uses the OpenSSH project to provide SSH. For example CentOS (another distro), there are two packages that need to be installed:

* openssh-client: the client
* openssh-server: the server side, only needed if you want someone to connect to you remotely

Both are typically installed already, but you can apt install them anyways to verify.



(This installed both the client and the server)

SSH is very useful, especially if you’re doing remote administration.

Text

Description automatically generated

Comments:

* Use the command ssh user@location to log into the machine remotely via ssh.
* **ssh** [**cstUser2@192.168.198.101**](mailto:cstUser2@192.168.198.101)
  + This logs into 192.168.198.101 (joe-nfsclient) as user cstUser2
  + It will then prompt for a password.
* Running the **who** command remotely will list all the users that are logged into joe-nfsclient.

We can do the same things from the NFS server -> client

Text

Description automatically generated

And, of course we can use hostnames as well, assuming we have the correct entries in the etc/hosts file.

Text

Description automatically generated\

Now we can ssh to the hostname instead of the IP address:

Text

Description automatically generated

Text

Description automatically generated

Once we’ve logged out of the remote machine, we’ll notice that there’s now a directory created call ~/.ssh. Inside that directory, you’ll find a file called known\_hosts – it’s got the fingerprint of the remote hosts that we’ve trusted so far. As part of the SSH protocol, the fingerprint will be compared to make sure you trust this machine.



# SCP

Besides logging into the remote machine, you can also copy files from the remote machine.

To copy a file **to** the remote machine:

* scp *file-to-copy remote-user@remote-host:new-filename*

To copy a file **from** the remote machine:

* scp *remote-user@remotehost:file-to-copy new-filename*

You will be prompted for a password associated with the account on the other computer.

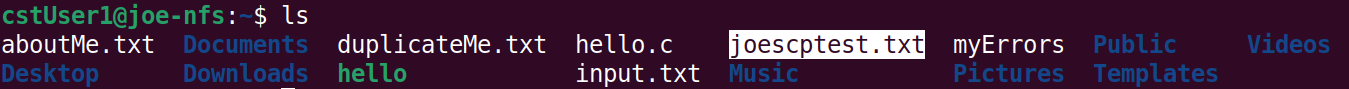
Transferring a file **to** a remote machine:

On the Client:

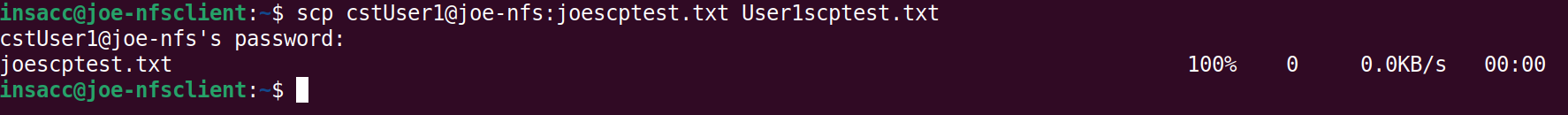
A picture containing text

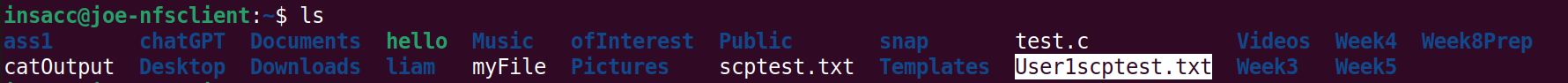
Description automatically generated

On the Server:



Transferring **from** a remote machine:





# Executing commands on a Remote Computer

Maybe you don’t want to log into the remote computer, you just want to execute a command. You can that too. Use the **ssh**  command, but include a command at the end.

For example, if you want to do an “ls”, you can execute the following: *ssh remote-user@hostname* ls

Text

Description automatically generated

Example: List the content of the /etc/fstab file on the remote machine:

Text

Description automatically generated

Comments:

* ssh cstUser1@joe-nfs cat /etc/fstab
  + This command logs into the nfs server (hostname for 192.168.198.100) as user **cstUser1**
  + Once complete, it executes the command “cat /etc/fstab” **as** catUser1 on the remote machine.
  + It takes the output of the command and displays it as the output of the ssh command.

It can get to be a bit of a hassle typing in passwords everytime. SSH understands public key encryptions (you’ve probably seen this in the math class). We can set up a public/private key, if you have the private key in the ~/.ssh directory of your local machine, and the public key in the ~/.ssh directory on the remote machine. The SSH protocol will then verify that you are a **trusted** person, and won’t ask for your password.

You can generate your public/private key pair with the command:

* ssh-keygen -t rsa

This uses the RSA algorithm to come up with your keypairs. Places the keys in the default location (~/.ssh/id\_rsa). Don’t specify a passphrase, unless you want to enter the passphrase every time the system accesses your key (we’re using these so that we don’t need to enter a password!)

Text

Description automatically generated

Your private key is saved in **~/.ssh/id\_rsa**, and your public key is saved in **~/.ssh/id\_rsa.pub**

Text

Description automatically generated with low confidence

Note the permission on the .ssh directory and the id\_rsa private key file, they are both not readable by group and others. This is for security; you don’t want anyone to ever get your private key (that’s why you could specify a passphrase).

Next we need to copy the public key to the remote computer that we don’t want to enter a password on. To do this, we would execute the following:

* ssh-copy-id *remoteuser@hostname*
  + The command “ssh-copy-id” copies the contents of the id\_rsa.pub file to the ~/.ssh/authorized\_key file on the server.

Text

Description automatically generated

Text

Description automatically generated

We can transfer files from our host machine (ie Windows) to our remote machine (client VM)

Graphical user interface, text, application

Description automatically generated

(Pardon the password typos. :/ )

