Name: Echiverri, Syd Ashley L	Date Performed:09/05/23
Course/Section:CPE 232-CPE31S4	Date Submitted:09/05/23
Instructor: Jonathan Taylar	Semester and SY: 2023-2024

Activity 2: SSH Key-Based Authentication and Setting up Git

1. **Objectives:**

1.1 Configure remote and local machine to connect via SSH using a KEY instead of using a password
1.2 Create a public key and private key
1.3 Verify connectivity
1.4 Setup Git Repository using local and remote repositories
1.5 Configure and Run ad hoc commands from local machine to remote servers

Part 1: Discussion

It is assumed that you are already done with the last Activity (**Activity 1: Configure Network using Virtual Machines).** *Provide screenshots for each task*.

It is also assumed that you have VMs running that you can SSH but requires a password. Our goal is to remotely login through SSH using a key without using a password. In this activity, we create a public and a private key. The private key resides in the local machine while the public key will be pushed to remote machines. Thus, instead of using a password, the local machine can connect automatically using SSH through an authorized key.

What Is ssh-keygen?

Ssh-keygen is a tool for creating new authentication key pairs for SSH. Such key pairs are used for automating logins, single sign-on, and for authenticating hosts.

SSH Keys and Public Key Authentication

The SSH protocol uses public key cryptography for authenticating hosts and users. The authentication keys, called SSH keys, are created using the keygen program.

SSH introduced public key authentication as a more secure alternative to the older .rhosts authentication. It improved security by avoiding the need to have password stored in files and eliminated the possibility of a compromised server stealing the user's password.

However, SSH keys are authentication credentials just like passwords. Thus, they must be managed somewhat analogously to usernames and passwords. They should have a proper termination process so that keys are removed when no longer needed.

Task 1: Create an SSH Key Pair for User Authentication

1. The simplest way to generate a key pair is to run *ssh-keygen* without arguments. In this case, it will prompt for the file in which to store keys. First,

the tool asked where to save the file. SSH keys for user authentication are usually stored in the users .ssh directory under the home directory. However, in enterprise environments, the location is often different. The default key file name depends on the algorithm, in this case *id_rsa* when using the default RSA algorithm. It could also be, for example, *id_dsa* or *id_ecdsa*.

```
sydechi@manageNode:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/sydechi/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/sydechi/.ssh/id_rsa.
Your public key has been saved in /home/sydechi/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:OIbpn/0BnFLhTsCHRXG8fnIw5DAU8aVYNIcjw+ZUds0 sydechi@manageNode
The key's randomart image is:
+---[RSA 2048]----+
|     ..*OBB.+o   |
|      ooXB=*  E  |
|       ==Boo     |
|     o *..=      |
|    o = S. o     |
|   . . o .o o    |
|    .     .+     |
|     . o   .     |
|      o ...      |
+----[SHA256]-----+
sydechi@manageNode:~$
```

2. Issue the command *ssh-keygen -t rsa -b 4096.* The algorithm is selected using the -t option and key size using the -b option.

3. When asked for a passphrase, just press enter. The passphrase is used for encrypting the key, so that it cannot be used even if someone obtains the private key file. The passphrase should be cryptographically strong.

```
sydechi@manageNode:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/sydechi/.ssh/id_rsa):
/home/sydechi/.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/sydechi/.ssh/id_rsa.
Your public key has been saved in /home/sydechi/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:SudGZFSgWmDyecOBnbFVJKAB/LA9Us/c4o8luLKmyss sydechi@manageNode
The key's randomart image is:
+---[RSA 4096]----+
| .o.+o++==+      |
|  o+o*o* .       |
|   *++B.o        |
|  o ++=+.        |
|   ..+..S        |
|    ..o=.        |
|     ..=o        |
|o o . ...        |
|+Eoo             |
+----[SHA256]-----+
```

4. Verify that you have created the key by issuing the command *ls -la .ssh.* The command should show the .ssh directory containing a pair of keys. For example, id_rsa.pub and id_rsa.

```
sydechi@manageNode:~$ ls -la .ssh
total 20
drwx------  2 sydechi sydechi 4096 Sep  5 16:49 .
drwxr-xr-x 16 sydechi sydechi 4096 Sep  5 16:39 ..
-rw-------  1 sydechi sydechi 3243 Sep  5 16:51 id_rsa
-rw-r--r--  1 sydechi sydechi  744 Sep  5 16:51 id_rsa.pub
-rw-r--r--  1 sydechi sydechi  888 Aug 15 18:08 known_hosts
sydechi@manageNode:~$
```

Task 2: Copying the Public Key to the remote servers

1. To use public key authentication, the public key must be copied to a server and installed in an *authorized_keys* file. This can be conveniently done using the *ssh-copy-id* tool.

```
sydechi@manageNode:~$ ssh-copy-id
Usage: /usr/bin/ssh-copy-id [-h|-?|-f|-n] [-i [identity_file]] [-p port] [[-o <
ssh -o options>] ...] [user@]hostname
        -f: force mode -- copy keys without trying to check if they are already
 installed
        -n: dry run    -- no keys are actually copied
        -h|-?: print this help
```

2. Issue the command similar to this: *ssh-copy-id -i ~/.ssh/id_rsa user@host*
3. Once the public key has been configured on the server, the server will allow any connecting user that has the private key to log in. During the login process, the client proves possession of the private key by digitally signing the key exchange.

```
sydechi@manageNode:~$ ssh-copy-id -i ~/.ssh/id_rsa sydechi@ControlNode1
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/sydechi/.s
sh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
 out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are promp
ted now it is to install the new keys
sydechi@controlnode1's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'sydechi@ControlNode1'"
and check to make sure that only the key(s) you wanted were added.
```

4. On the local machine, verify that you can SSH with Server 1 and Server 2. What did you notice? Did the connection ask for a password? If not, why?

Server 1:

```
sydechi@manageNode:~$ ssh-copy-id -i ~/.ssh/id_rsa sydechi@ControlNode1
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/sydechi/.s
sh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
 out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are promp
ted now it is to install the new keys
sydechi@controlnode1's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'sydechi@ControlNode1'"
and check to make sure that only the key(s) you wanted were added.
```

Server 2:

```
sydechi@manageNode:~$ ssh-copy-id -i ~/.ssh/id_rsa sydechi@ControlNode2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/sydechi/.s
sh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
 out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are promp
ted now it is to install the new keys
sydechi@controlnode2's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'sydechi@ControlNode2'"
and check to make sure that only the key(s) you wanted were added.
```

I discovered that using the command "ssh-copy-id" is quite useful when entering servers since it does not require you to enter your account and password every time. You attempt to connect to servers. The command is useful because it can identify the user as an authorized user with full access to the server. It works because we added a key to the server, and the only one who has unfettered access to it is the main server. the local machine.

Reflections:

Answer the following:

1. How will you describe the ssh-program? What does it do?
   SSH is a crucial tool for managing and securing remote connections to servers, providing encryption, access, and authentication. It is essential for data transfer, system administration, and secure remote server administration, making it essential for system administrators.

2. How do you know that you already installed the public key to the remote servers?

   The public key was installed on distant servers and added to specific servers without prompting login or password. It can be identified by not prompting login or password. The key can be found in the /.ssh/authorized_keys file.

Part 2: Discussion

Provide screenshots for each task.

It is assumed that you are done with the last activity (**Activity 2: SSH Key-Based Authentication**).

Set up Git

At the heart of GitHub is an open-source version control system (VCS) called Git. Git is responsible for everything GitHub-related that happens locally on your computer. To use Git on the command line, you'll need to download, install, and configure Git on your computer. You can also install GitHub CLI to use GitHub from the command line. If you don't need to work with files locally, GitHub lets you complete many Git-related actions directly in the browser, including:

- Creating a repository
- Forking a repository
- Managing files
- Being social

Task 3: Set up the Git Repository

1. On the local machine, verify the version of your git using the command *which git*. If a directory of git is displayed, then you don't need to install git. Otherwise, to install git, use the following command: *sudo apt install git*

```
sydechi@ControlNode1:~$ sudo apt install git
[sudo] password for sydechi:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libllvm7
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk
  gitweb git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 4,817 kB of archives.
After this operation, 34.3 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ph.archive.ubuntu.com/ubuntu bionic/main amd64 liberror-perl all 0
.17025-1 [22.8 kB]
Get:2 http://ph.archive.ubuntu.com/ubuntu bionic-updates/main amd64 git-man all
 1:2.17.1-1ubuntu0.18 [804 kB]
Get:3 http://ph.archive.ubuntu.com/ubuntu bionic-updates/main amd64 git amd64 1
:2.17.1-1ubuntu0.18 [3,990 kB]
Fetched 4,817 kB in 5s (1,065 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 170157 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17025-1_all.deb ...
```

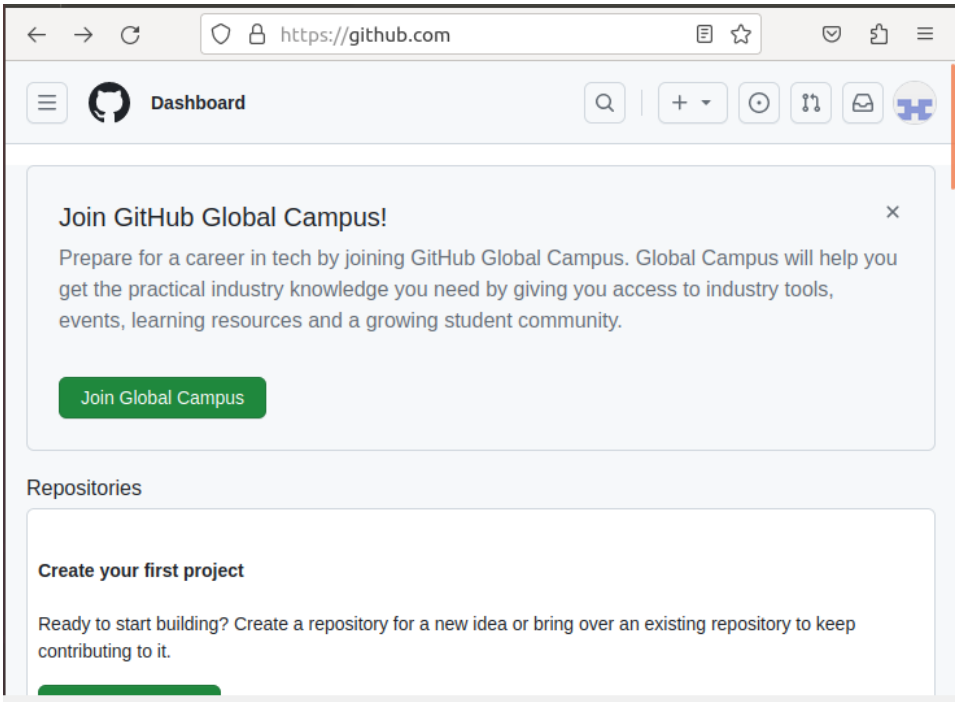
2. After the installation, issue the command *which git* again. The directory of git is usually installed in this location: *user/bin/git*.

```
sydechi@ControlNode1:~$ which git
/usr/bin/git
```

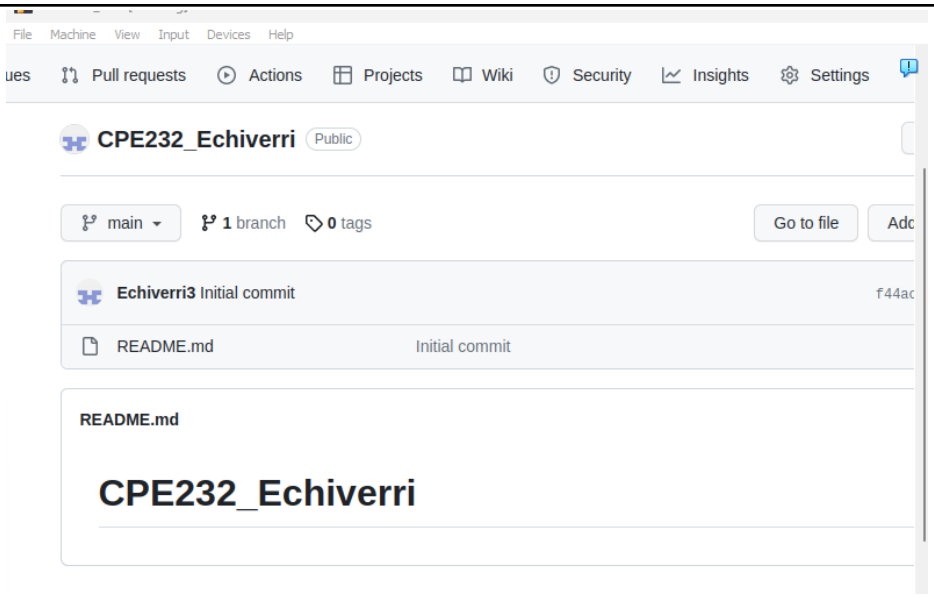
3. The version of git installed in your device is the latest. Try issuing the command *git --version* to know the version installed.

```
sydechi@ControlNode1:~$ git --version
git version 2.17.1
```

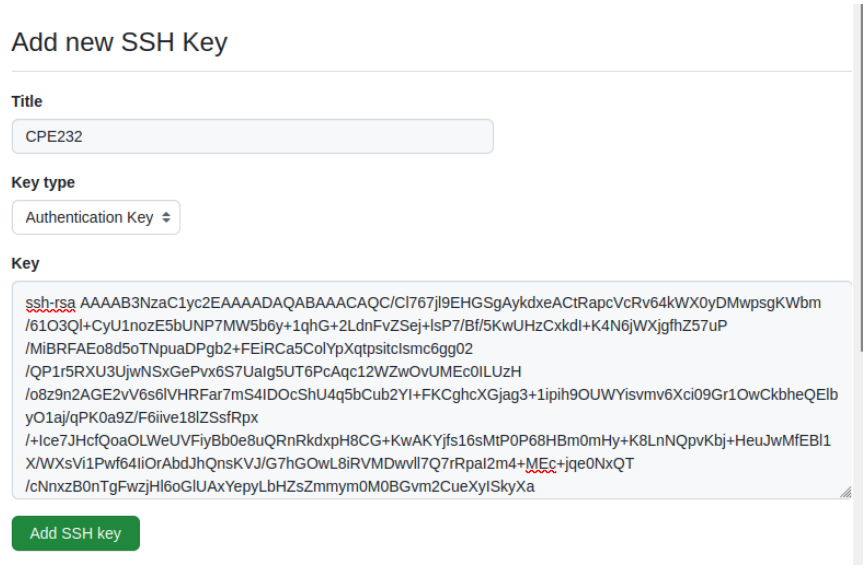
4. Using the browser in the local machine, go to www.github.com.



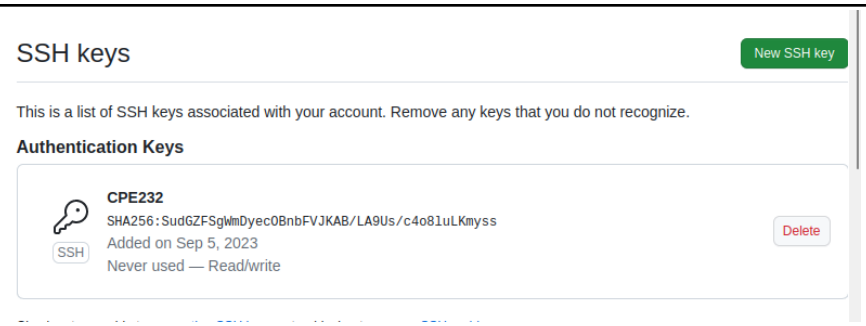
5. Sign up in case you don't have an account yet. Otherwise, login to your GitHub account.
    a. Create a new repository and name it as CPE232_yourname. Check Add a README file and click Create repository.



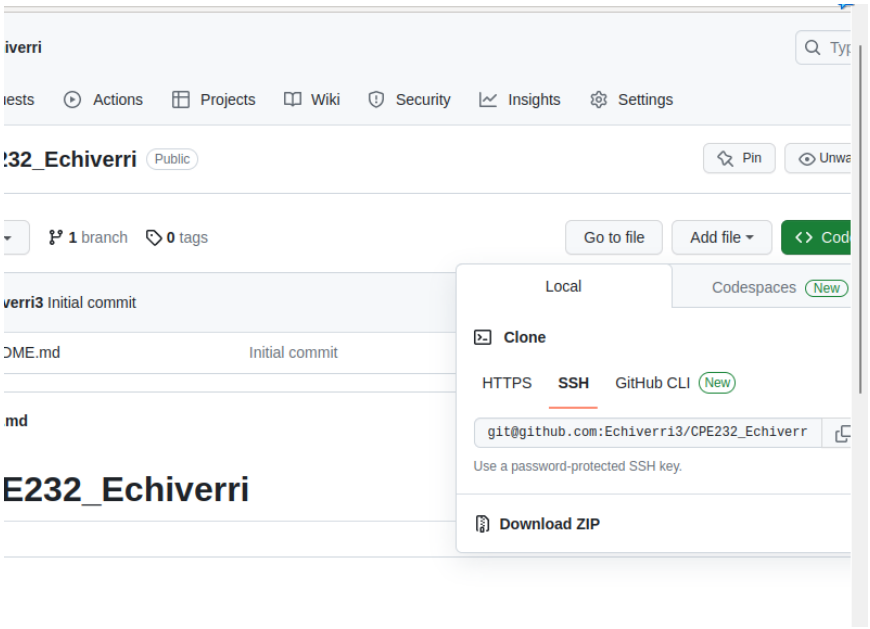
b. Create a new SSH key on GitHub. Go your profile's setting and click SSH and GPG keys. If there is an existing key, make sure to delete it. To create a new SSH keys, click New SSH Key. Write CPE232 key as the title of the key.



c. On the local machine's terminal, issue the command cat .ssh/id_rsa.pub and copy the public key. Paste it on the GitHub key and press Add SSH key.



d. Clone the repository that you created. In doing this, you need to get the link from GitHub. Browse to your repository as shown below. Click on the Code drop down menu. Select SSH and copy the link.



e. Issue the command git clone followed by the copied link. For example, *git clone* git@github.com:jvtaylar-cpe/CPE232_yourname.git. When prompted to continue connecting, type yes and press enter.

```
sydechi@manageNode:~$ git clone git@github.com:Echiverri3/CPE232_Echiverri.git
Cloning into 'CPE232_Echiverri'...
The authenticity of host 'github.com (20.87.225.212)' can't be established.
ECDSA key fingerprint is SHA256:p2QAMXNIC1TJYWeIOttrVc98/R1BUFWu3/LiyKgUfQM.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added 'github.com,20.87.225.212' (ECDSA) to the list of kno
wn hosts.
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

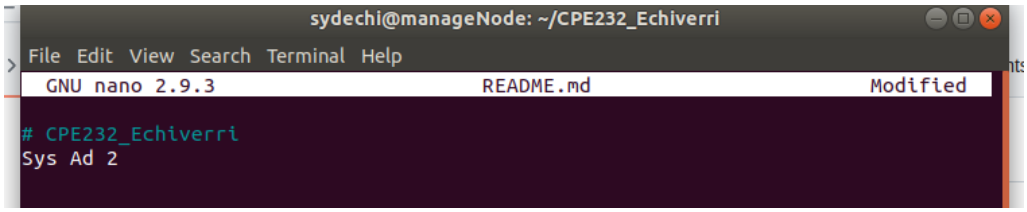
f. To verify that you have cloned the GitHub repository, issue the command *ls*. Observe that you have the CPE232_yourname in the list of your directories. Use CD command to go to that directory and LS command to see the file README.md.

```
sydechi@manageNode:~$ ls
CPE232_Echiverri  Documents  examples.desktop  Pictures  Templates
Desktop           Downloads  Music             Public    Videos
sydechi@manageNode:~$ cd CPE232_Echiverri
sydechi@manageNode:~/CPE232_Echiverri$ ls
README.md
sydechi@manageNode:~/CPE232_Echiverri$
```

g. Use the following commands to personalize your git.
   - *git config --global user.name "Your Name"*
   - *git config --global user.email yourname@email.com*
   - Verify that you have personalized the config file using the command *cat ~/.gitconfig*

```
sydechi@manageNode:~/CPE232_Echiverri$ git config --global user.name "Echi"
sydechi@manageNode:~/CPE232_Echiverri$ git config --global user.email "qsaechive
rri@tip.edu.ph"
sydechi@manageNode:~/CPE232_Echiverri$ cat ~/.gitconfig
[user]
        name = Echi
        email = qsaechiverri@tip.edu.ph
sydechi@manageNode:~/CPE232_Echiverri$
```

h. Edit the README.md file using nano command. Provide any information on the markdown file pertaining to the repository you created. Make sure to write out or save the file and exit.



i. Use the *git status* command to display the state of the working directory and the staging area. This command shows which changes have been staged, which haven't, and which files aren't being tracked by Git. Status output does not show any information regarding the committed project history. What is the result of issuing this command?

```
sydechi@manageNode:~/CPE232_Echiverri$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
sydechi@manageNode:~/CPE232_Echiverri$
```

j. Use the command *git add README.md* to add the file into the staging area.

```
sydechi@manageNode:~/CPE232_Echiverri$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

sydechi@manageNode:~/CPE232_Echiverri$
```

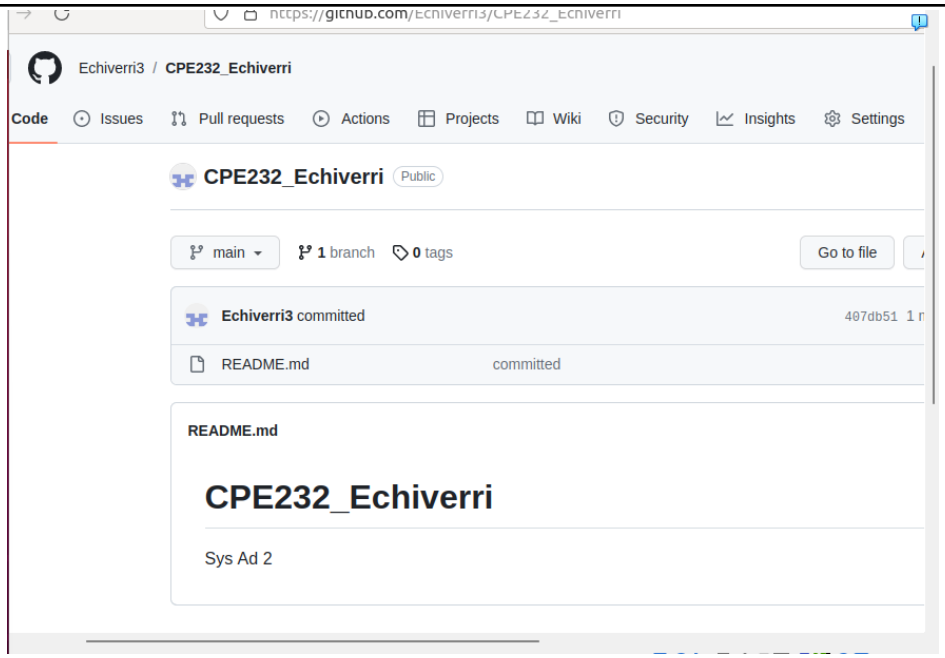
k. Use the *git commit -m "your message"* to create a snapshot of the staged changes along the timeline of the Git projects history. The use of this command is required to select the changes that will be staged for the next commit.

```
sydechi@manageNode:~/CPE232_Echiverri$ git commit -m "committed"
[main 407db51] committed
 1 file changed, 2 insertions(+), 1 deletion(-)
sydechi@manageNode:~/CPE232_Echiverri$
```

l. Use the command *git push <remote><branch>* to upload the local repository content to GitHub repository. Pushing means to transfer commits from the local repository to the remote repository. As an example, you may issue *git push origin main.*

```
sydechi@manageNode:~/CPE232_Echiverri$ git push
Counting objects: 3, done.
Writing objects: 100% (3/3), 265 bytes | 265.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:Echiverri3/CPE232_Echiverri.git
   f44ac47..407db51  main -> main
sydechi@manageNode:~/CPE232_Echiverri$
```

m. On the GitHub repository, verify that the changes have been made to README.md by refreshing the page. Describe the README.md file. You can notice the how long was the last commit. It should be some minutes ago and the message you typed on the git commit command should be there. Also, the README.md file should have been edited according to the text you wrote.



Reflections:

Answer the following:

3. What sort of things have we so far done to the remote servers using ansible commands?

   An SSH Key Pair was created for user authentication, storing the encryption key and copying the public key to remote servers. This allowed the main system to access distant servers without prompting the user, enabling manipulation and testing.

4. How important is the inventory file?

   The inventory file acts as a blueprint, detailing how individual hosts and host groups interact with commands, modules, and tasks in a playbook. The format of this file varies depending on the configuration and plugins in use in your environment.

Conclusions/Learnings:

This activity explained the usage of SSH keys for remote servers, the creation of public and private keys, encryption, and the use of Git commands. It also illustrated how to use a CLI to connect to a website and add, commit, and push files to a repository. The goals were to connect remote and local workstations through SSH, generate public and private keys, check connectivity, set up Git repositories, and configure ad hoc commands from local to remote servers.