Name: De Omampo, Julius Mark A.	**Date Performed:** 09/04/24
Course/Section: CPE212– CPE31S2	**Date Submitted:** 09/04/24
Instructor: Engr. Robin Valenzuela	**Semester and SY:** 1st Sem. 2024 - 2025

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.

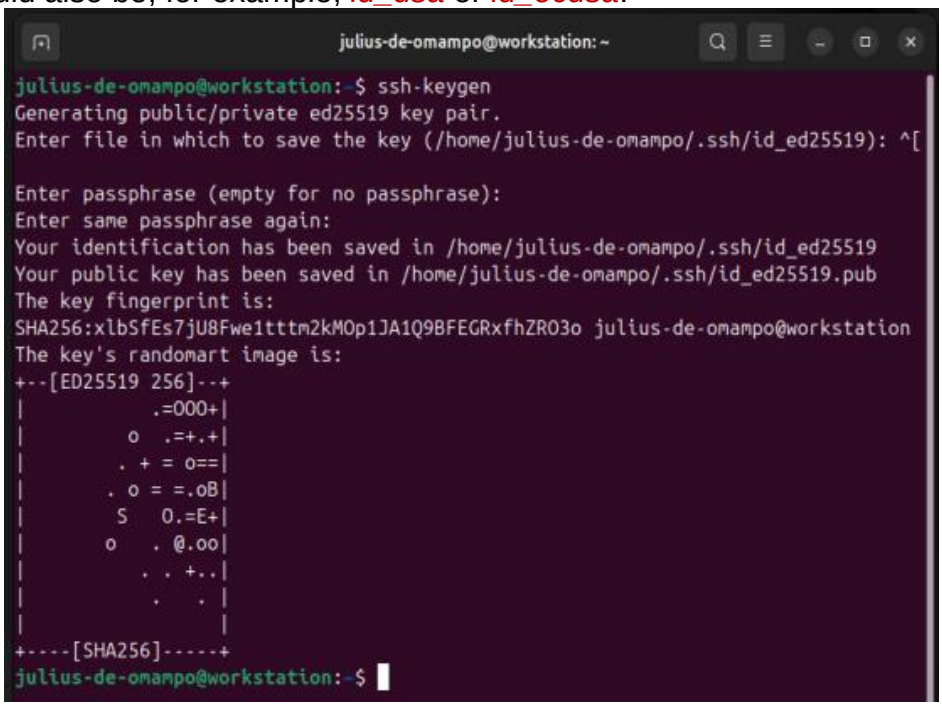
SSH introduced public key authentication as a more secure alternative to the older .rhosts authentication. It improved security by avoiding the need to have password stored in files and eliminated the possibility of a compromised server stealing the user's password.

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.

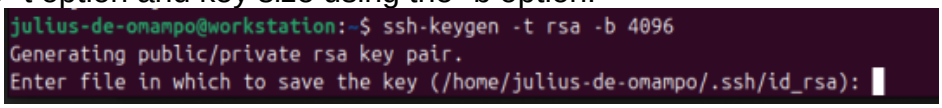
Task 1: Create an SSH Key Pair for User Authentication

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

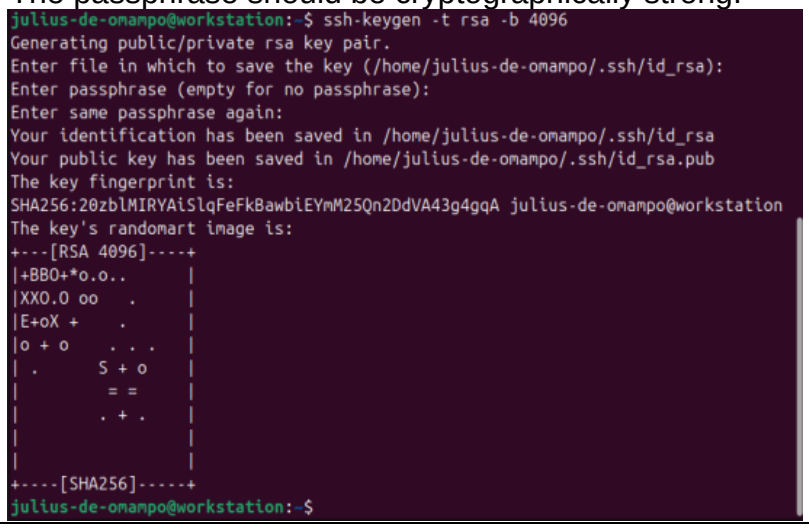
where to save the file. SSH keys for user authentication are usually stored in the users .ssh directory under the home directory. However, in enterprise environments, the location is often different. The default key file name depends on the algorithm, in this case *id_rsa* when using the default RSA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



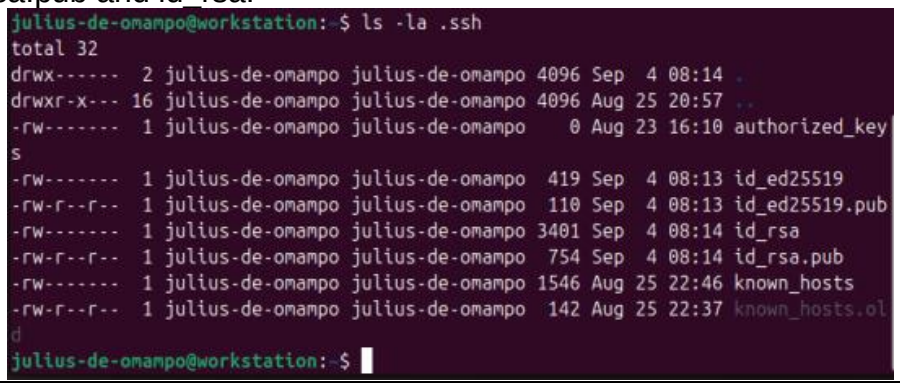
2. Issue the command *ssh-keygen -t rsa -b 4096.* The algorithm is selected using the -t option and key size using the -b option.



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.



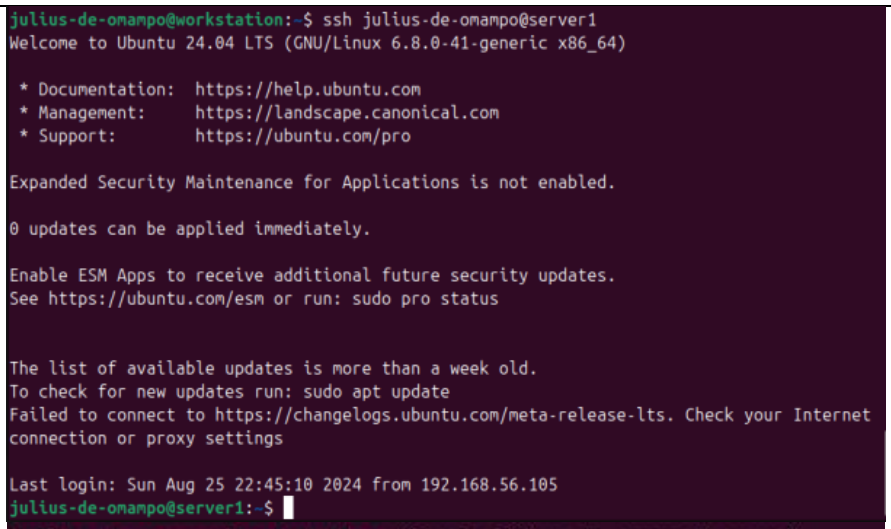
4. Verify that you have created the key by issuing the command *ls -la .ssh.* The command should show the .ssh directory containing a pair of keys. For example, id_rsa.pub and id_rsa.

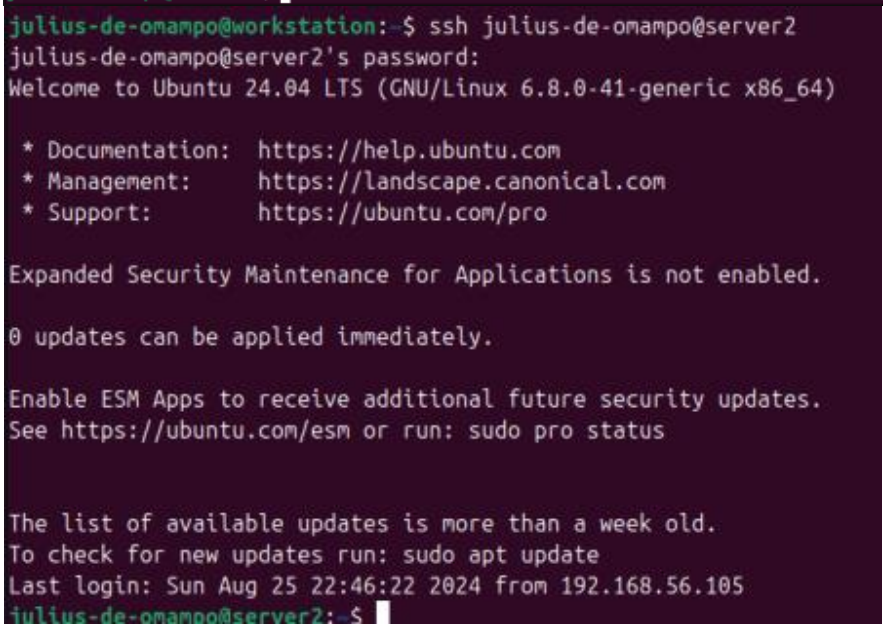


Task 2: Copying the Public Key to the remote servers

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.
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.
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?





When connecting to server1 from the local machine, it didn't prompt me for a password, whilst connecting to server2 did. This is because I only copied the SSH keys from server1 and since I didn't copy the SSH keys from server2, accessing server2 through SSH will certainly prompt me for a password.

Reflections:

Answer the following:

1. How will you describe the ssh-program? What does it do?

From the said activity, we had basically simulated how to create SSH keys in various devices, demonstrate how to copy it and implement it when accessing the said devices remotely. Copying SSH keys is greatly beneficial for supervision and administration, as it eliminates the redundancy of entering a password when accessing a remote device via SSH.

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

You can check if you have already installed the public key if you open the *authorized_keys* file within the .ssh directory of the remote server, you will see the and verify if it's installed if it shows the *user@hostname* of the device which copied the servers' SSH key. You could also check it from the local machine if you access the server remotely via SSH and it didn't prompt you a password, and through that, you could check the servers' authorized keys by issuing the *cat ~/.ssh/authorized_keys* command.

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*

```
julius-de-omampo@workstation:~$ sudo apt install git
[sudo] password for julius-de-omampo:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc 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 11 not upgraded.
Need to get 4,804 kB of archives.
After this operation, 24.5 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

```
Get:1 http://ph.archive.ubuntu.com/ubuntu noble/main amd64 liberror-perl all 0.17029-2 [25
.6 kB]
Get:2 http://ph.archive.ubuntu.com/ubuntu noble-updates/main amd64 git-man all 1:2.43.0-1u
buntu7.1 [1,100 kB]
Get:3 http://ph.archive.ubuntu.com/ubuntu noble-updates/main amd64 git amd64 1:2.43.0-1ubu
ntu7.1 [3,679 kB]
Fetched 4,804 kB in 3s (1,874 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 150835 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17029-2_all.deb ...
Unpacking liberror-perl (0.17029-2) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1%3a2.43.0-1ubuntu7.1_all.deb ...
Unpacking git-man (1:2.43.0-1ubuntu7.1) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1%3a2.43.0-1ubuntu7.1_amd64.deb ...
Unpacking git (1:2.43.0-1ubuntu7.1) ...
Setting up liberror-perl (0.17029-2) ...
Setting up git-man (1:2.43.0-1ubuntu7.1) ...
Setting up git (1:2.43.0-1ubuntu7.1) ...
Processing triggers for man-db (2.12.0-4build2) ...
julius-de-omampo@workstation:~$
```

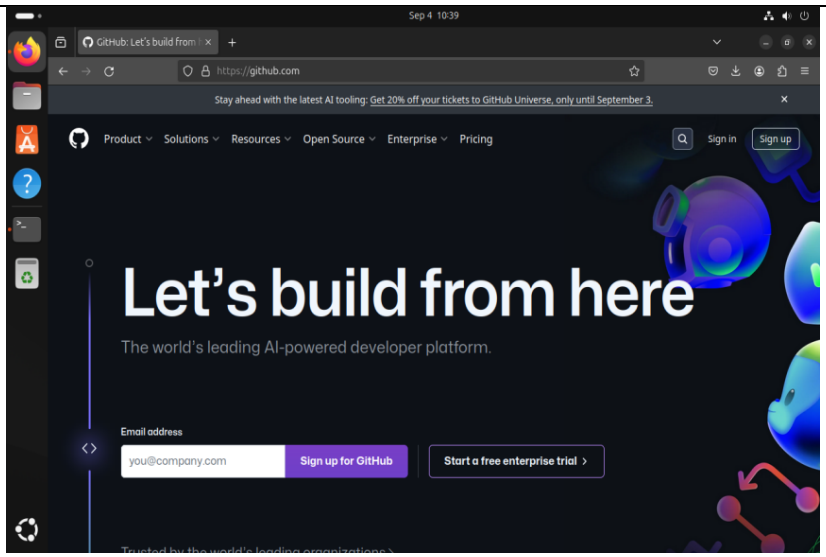
2. After the installation, issue the command *which git* again. The directory of git is usually installed in this location: *user/bin/git.*

```
julius-de-omampo@workstation:~$ which git
/usr/bin/git
```

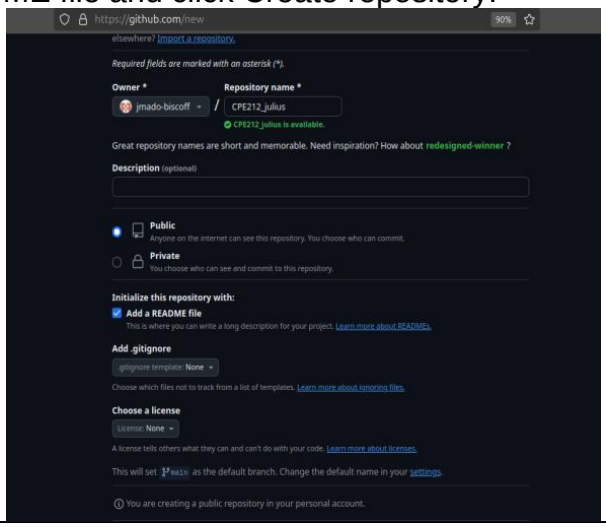
3. The version of git installed in your device is the latest. Try issuing the command *git --version* to know the version installed.

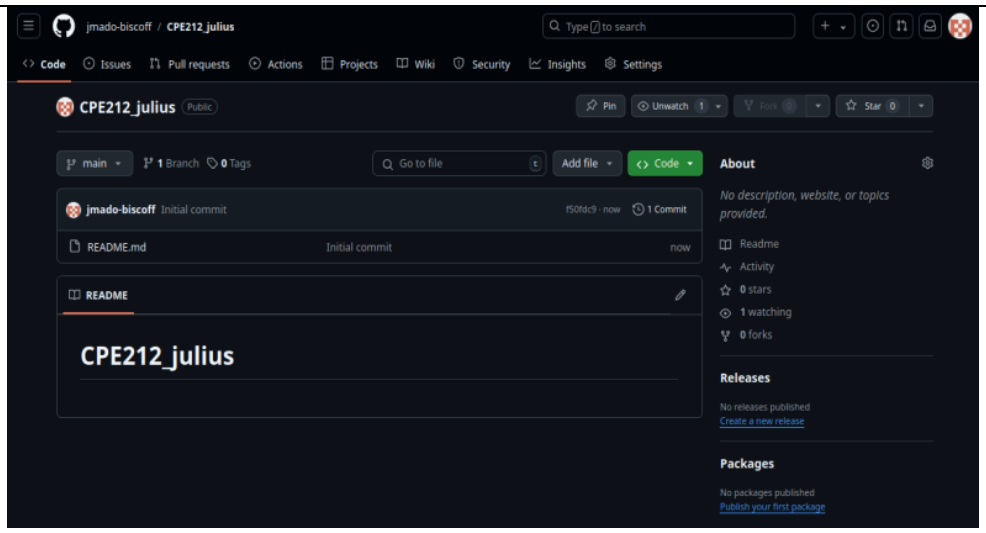
```
julius-de-omampo@workstation:~$ git --version
git version 2.43.0
```

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

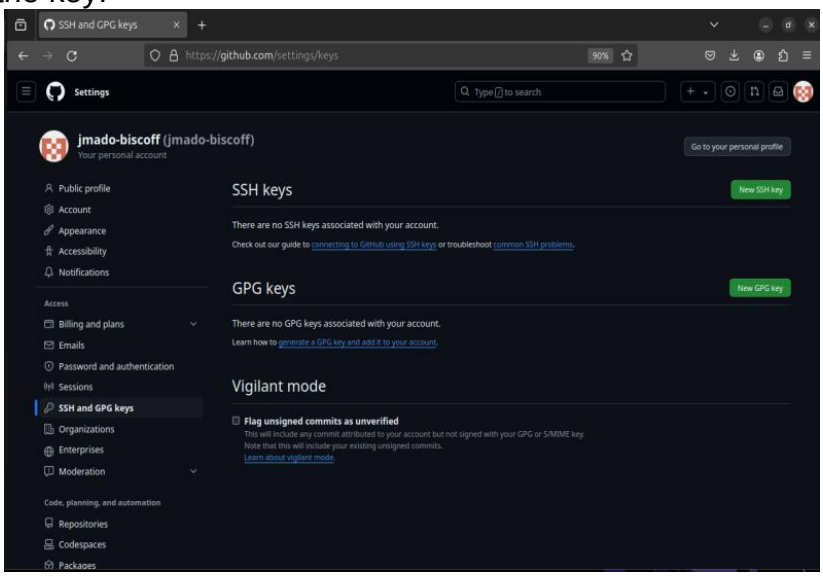


5. Sign up in case you don't have an account yet. Otherwise, login to your GitHub account.
    a. Create a new repository and name it as CPE212_yourname. Check Add a README file and click Create repository.

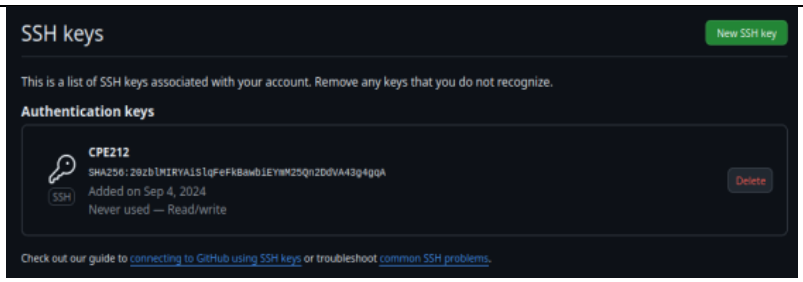




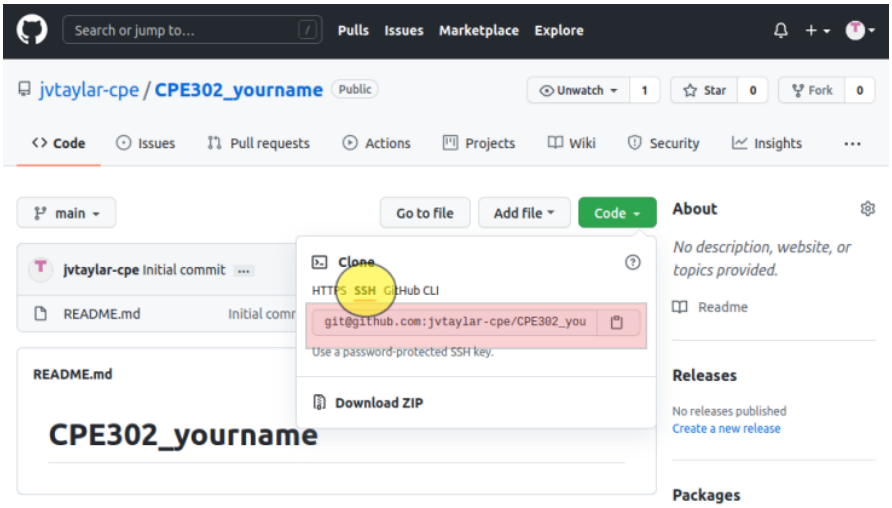
b. Create a new SSH key on GitHub. Go your profile's setting and click SSH and GPG keys. If there is an existing key, make sure to delete it. To create a new SSH keys, click New SSH Key. Write CPE212 key as the title of the key.

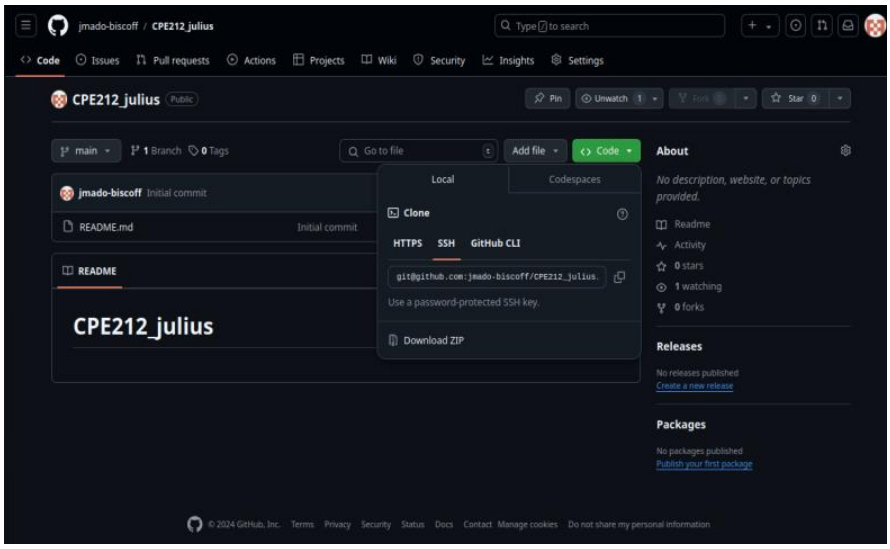


c. On the local machine's terminal, issue the command cat .ssh/id_rsa.pub and copy the public key. Paste it on the GitHub key and press Add SSH key.



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/CPE212_yourname.git*. When prompted to continue connecting, type yes and press enter.

```
julius-de-omampo@workstation:~$ git clone git@github.com:jmado-biscoff/CPE212_julius.git
Cloning into 'CPE212_julius'...
The authenticity of host 'github.com (20.205.243.166)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added 'github.com' (ED25519) to the list of known 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 (from 0)
Receiving objects: 100% (3/3), done.
julius-de-omampo@workstation:~$
```

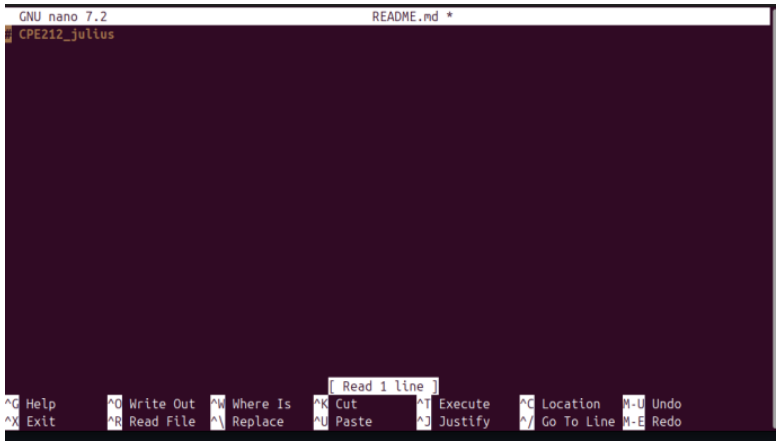
f. To verify that you have cloned the GitHub repository, issue the command *ls*. Observe that you have the CPE212_yourname in the list of your directories. Use CD command to go to that directory and LS command to see the file README.md.

```
julius-de-omampo@workstation:~$ ls
CPE212_julius  Documents  Music     Public  Templates
Desktop        Downloads  Pictures  snap    Videos
julius-de-omampo@workstation:~$ cd CPE212_julius/
julius-de-omampo@workstation:~/CPE212_julius$ ls
README.md
```

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*

```
julius-de-omampo@workstation:~/CPE212_julius$ git config --global user.name "jmado_biscoff"
julius-de-omampo@workstation:~/CPE212_julius$ git config --global user.email "qjmadeomampo@tip.edu.ph"
julius-de-omampo@workstation:~/CPE212_julius$ cat ~/.gitconfig
[user]
        name = jmado_biscoff
        email = qjmadeomampo@tip.edu.ph
julius-de-omampo@workstation:~/CPE212_julius$
```

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?

```
julius-de-omampo@workstation:~/CPE212_julius$ 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 restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
julius-de-omampo@workstation:~/CPE212_julius$
```

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

```
julius-de-omampo@workstation:~/CPE212_julius$ git add README.md
```

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.

```
julius-de-omampo@workstation:~/CPE212_julius$ git commit -m "hello world"
[main 55a3a87] hello world
 1 file changed, 1 insertion(+), 1 deletion(-)
```

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 pugish origin main.*

```
julius-de-omampo@workstation:~/CPE212_julius$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 263 bytes | 263.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:jmado-biscoff/CPE212_julius.git
   f50fdc9..55a3a87  main -> main
julius-de-omampo@workstation:~/CPE212_julius$
```

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?

4. How important is the inventory file?

Conclusions/Learnings:

In this experiment, I have learned about the various applications of SSH, in connecting to a remote device/server and linking GitHub Repositories to local terminal codes. SSH serves a great role for delivering packages and commands from a client to another. It is great as it eliminates the need of travel of going to one place to another just to modify, update, and maintain a specific device, you can use SSH to remotely access them. Moreover, SSH is a reliable and efficient tool for remote access, as it possesses secure encryption, authentication, and keys. Overall, SSH is a great tool for administering remote devices and for delivering codes and information to online repositories.