Name: Ian Carlo T. Bello	**Date Performed:** September 1, 2022
Course/Section: CPE232 – CPE31S24	**Date Submitted:** September 1, 2022
Instructor: Dr. Jonathan V. Taylar	**Semester and SY:** 1st Sem – 3rd Year

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.

```
ubuntuhost@workstation:~$ ssh-keygen -t rsa -b 4906
Generating public/private rsa key pair.
```

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.

```
ubuntuhost@workstation:~$ ssh-keygen -t rsa -b 4906
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntuhost/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntuhost/.ssh/id_rsa
Your public key has been saved in /home/ubuntuhost/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:Q6SmvnXwkC473Wixiwa89ui7rPrSCa6GtWxXbg4EiG8 ubuntuhost@workstation
The key's randomart image is:
+---[RSA 4906]----+
|        .        |
|..     o         |
|o .   o .        |
| . . o o         |
| .E o + S        |
|..+o .o+ .       |
|o= =++o=o        |
|o+O.+BB..        |
|BBB*==o.         |
+----[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.

```
ubuntuhost@workstation:~$ ls -la .ssh
total 24
drwx------  2 ubuntuhost ubuntuhost 4096 Sep  1 15:30 .
drwxr-x--- 16 ubuntuhost ubuntuhost 4096 Sep  1 15:26 ..
-rw-------  1 ubuntuhost ubuntuhost 4002 Sep  1 15:30 id_rsa
-rw-r--r--  1 ubuntuhost ubuntuhost  880 Sep  1 15:30 id_rsa.pub
-rw-------  1 ubuntuhost ubuntuhost 2240 Aug 24 19:35 known_hosts
-rw-------  1 ubuntuhost ubuntuhost 1120 Aug 24 19:30 known_hosts.old
ubuntuhost@workstation:~$
```

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*

```
ubuntuhost@workstation:~$ ssh-copy-id -i ~/.ssh/id_rsa ubuntuhost@server1
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/ubuntuhost
/.ssh/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
ubuntuhost@server1's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'ubuntuhost@server1'"
and check to make sure that only the key(s) you wanted were added.

ubuntuhost@workstation:~$ ssh-copy-id -i ~/.ssh/id_rsa ubuntuhost@server2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/ubuntuhost
/.ssh/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
ubuntuhost@server2's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'ubuntuhost@server2'"
and check to make sure that only the key(s) you wanted were added.

ubuntuhost@workstation:~$
```

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?
   I didn’t ask for their password because we transferred a public key.

```
ubuntuhost@workstation:~$ ssh ubuntuhost@server1
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-46-generic x86_64)

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

0 updates can be applied immediately.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Last login: Thu Sep  1 15:51:11 2022 from 192.168.56.102
ubuntuhost@server1:~$
logout
Connection to server1 closed.
ubuntuhost@workstation:~$ ssh ubuntuhost@server2
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-46-generic x86_64)

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

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

*** System restart required ***
Last login: Wed Aug 24 19:35:59 2022 from 192.168.56.102
ubuntuhost@server2:~$
```

Reflections:

Answer the following:

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

It is easier to connect, and it does have security and encrypting better than telnet.

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

After transferring the public key using copy-id command we can already tell by connecting we didn't have a prompt for password login.

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*

```
ubuntuhost@workstation:~$ sudo apt install git
[sudo] password for ubuntuhost:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are
  chromium-codecs-ffmpeg-extra gstreamer1.0-vaapi
  libgstreamer-plugins-bad1.0-0 libva-wayland2
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-email gi
```

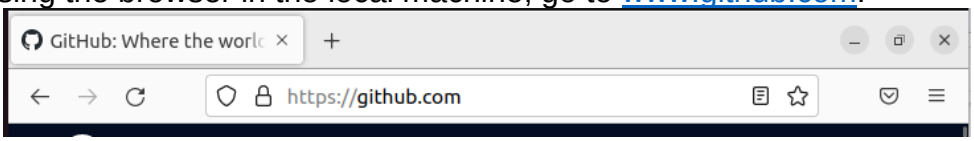
2. After the installation, issue the command *which git* again. The directory of git is usually installed in this location: *user/bin/git.*

```
ubuntuhost@workstation:~$ which git
/usr/bin/git
ubuntuhost@workstation:~$
```

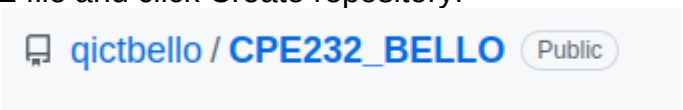
3. The version of git installed in your device is the latest. Try issuing the command *git --version* to know the version installed.

```
ubuntuhost@workstation:~$ git --version
git version 2.34.1
ubuntuhost@workstation:~$
```

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

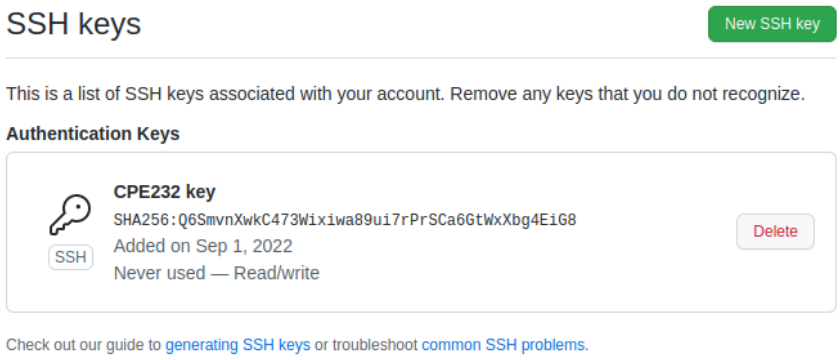


5. Sign up in case you don't have an account yet. Otherwise, login to your GitHub account.
   a. Create a new repository and name it as CPE232_yourname. Check Add a README file and click Create repository.

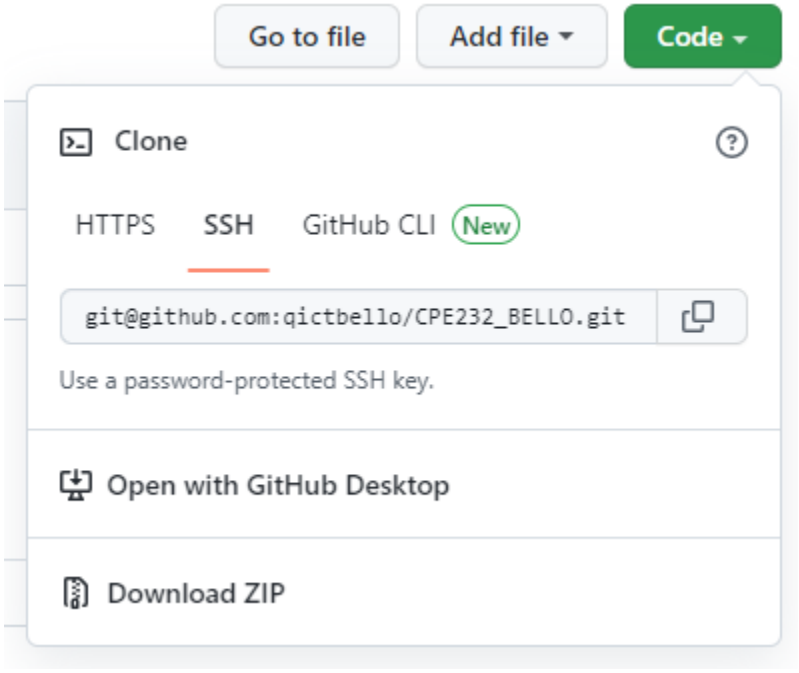


b. Create a new SSH key on GitHub. Go your profile's setting and click SSH and GPG keys. If there is an existing key, make sure to delete it. To create a new SSH keys, click New SSH Key. Write CPE232 key as the title of the key.

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.

```
ubuntuhost@workstation:~$ git clone git@github.com:qictbello/CPE232_BELLO.git
Cloning into 'CPE232_BELLO'...
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])? 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
Receiving objects: 100% (3/3), done.
ubuntuhost@workstation:~$
```

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.

```
ubuntuhost@workstation:~$ ls
CPE232_BELLO  Documents  id_rsa      Music     Public  Templates
Desktop       Downloads  id_rsa.pub  Pictures  snap    Videos
ubuntuhost@workstation:~$ cd CPE232_BELLO
ubuntuhost@workstation:~/CPE232_BELLO$ ls
README.md
ubuntuhost@workstation:~/CPE232_BELLO$
```

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*

```
ubuntuhost@workstation:~/CPE232_BELLO$ git config --global user.name "Bello"
ubuntuhost@workstation:~/CPE232_BELLO$ git config --global user.email qictbello
@tip.edu.ph
ubuntuhost@workstation:~/CPE232_BELLO$ cat ~/.gitconfig
[user]
        name = Bello
        email = qictbello@tip.edu.ph
ubuntuhost@workstation:~/CPE232_BELLO$
```

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.

```
  GNU nano 6.2
# CPE232_BELLO


#cute si bello
```

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?
It shows that the readme.md file is modified.

```
ubuntuhost@workstation:~/CPE232_BELLO$ 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")
ubuntuhost@workstation:~/CPE232_BELLO$
```

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

```
ubuntuhost@workstation:~/CPE232_BELLO$ git add README.md
```

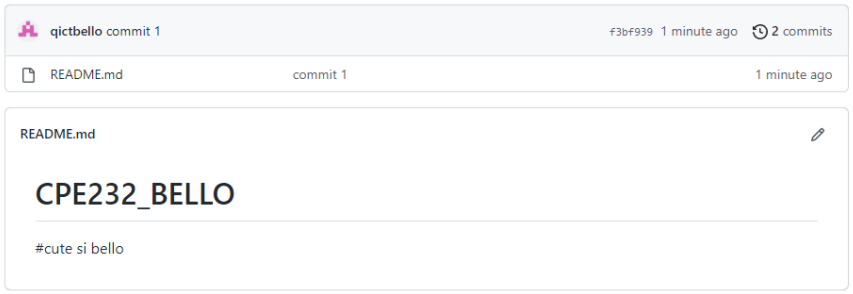
k. Use the *git commit -m "your message"* to create a snapshot of the staged changes along the timeline of the Git projects history. The use of this command is required to select the changes that will be staged for the next commit.

```
ubuntuhost@workstation:~/CPE232_BELLO$ git commit -m "commit 1"
[main f3bf939] commit 1
 1 file changed, 4 insertions(+), 1 deletion(-)
ubuntuhost@workstation:~/CPE232_BELLO$
```

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.*

```
ubuntuhost@workstation:~/CPE232_BELLO$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 266 bytes | 266.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:qictbello/CPE232_BELLO.git
   cc3d98a..f3bf939  main -> main
ubuntuhost@workstation:~/CPE232_BELLO$
```

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? **We connected our created key into our GitHub and created repository that is shared between GitHub and the terminal/local machine. We also configure and made a change in the readme.md file.**
4. How important is the inventory file?
   This is our place where we can work around, we can share keys and have other people modify projects and files with this setup. This repository would serve as our group shared file if we gave other access with our key.

Conclusions/Learnings:

In conclusion, we learned about how to connect our public and private keys to other hosts and servers. We did have access to both servers without a password using the keys, and we used these keys to have our repository on GitHub, where we can share our repository as our inventory of files. We learned about git commands that are useful for modifying files, such as add, commit, and pushing them into the GitHub repository. This activity will be helpful in managing servers and files.