Name:Jessie Robert Lazo	Date Performed:9/11/2024
Course/Section:CPE31S2	Date Submitted:9/11/2024
Instructor: Engr. Robin Valenzuela	Semester and SY:

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*.

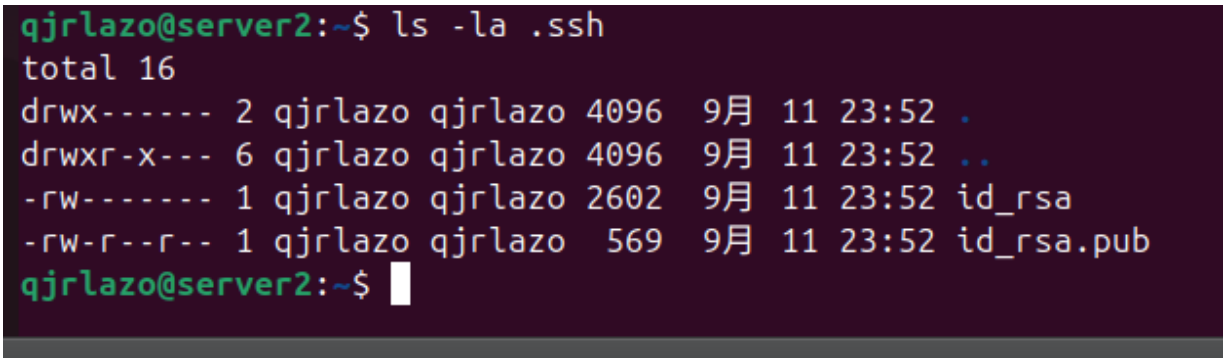
```
qjrlazo@server2:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/qjrlazo/.ssh/id_rsa):
Created directory '/home/qjrlazo/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/qjrlazo/.ssh/id_rsa
Your public key has been saved in /home/qjrlazo/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:9jF75QDy9htIohS0l0WfYQC1tTwxAPzwXrziMgZGBXQ qjrlazo@server2
The key's randomart image is:
+---[RSA 3072]----+
|     .+oE=*o*    |
|     . o+o * *   |
|      o.++o.*    |
|      .o oo.o.   |
|     .. S.*....  |
|     .oo =o*.+   |
|     ... .+.+ .  |
|        + .. o   |
|       . o  .    |
+----[SHA256]-----+
qjrlazo@server2:~$
qjrlazo@server2:~$
```

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.

```
qjrlazo@server2:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/qjrlazo/.ssh/id_rsa):
/home/qjrlazo/.ssh/id_rsa already exists.
Overwrite (y/n)?
qjrlazo@server2:~$ S
```

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?
   - It asks for a password, since ssh is successful.

Server1:

```
qjrlazo@workstation:~$ ssh qjrlazo@server1
qjrlazo@server1's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-40-generic x86_64)

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

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

Last login: Wed Sep 11 23:42:31 2024 from 192.168.56.103
qjrlazo@server1:~$
```

Server2:

```
qjrlazo@workstation:~$ ssh qjrlazo@server2
qjrlazo@server2's password:
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.8.0-40-generic x86_64)

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

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

Last login: Wed Sep 11 23:46:19 2024 from 192.168.56.103
qjrlazo@server2:~$
```

Reflections:

Answer the following:

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

The `ssh` (Secure Shell) program is a network protocol and tool used to securely access and manage remote systems over an unsecured network. It provides encrypted communication between a client and a server, allowing users to execute commands, transfer files, and manage remote systems safely.

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

I was able to login in server 1 and 2 when I try ssh qjrlazo@server1 and ssh qjrlazo@server2

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

Installation:

```
qjrlazo@workstation:~$ sudo apt install git
[sudo] password for qjrlazo:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  acpi-support acpid aisleriot apturl apturl-common attr branding-ubuntu cheese cheese-common endeavour
  endeavour-common fonts-kacst fonts-kacst-one fonts-khmeros-core fonts-lao fonts-liberation2 fonts-lklug-sinhala
  fonts-sil-abyssinica fonts-sil-padauk fonts-thai-tlwg fonts-tibetan-machine fonts-tlwg-garuda fonts-tlwg-garuda-ttf
  fonts-tlwg-kinnari fonts-tlwg-kinnari-ttf fonts-tlwg-laksaman fonts-tlwg-laksaman-ttf fonts-tlwg-loma
  fonts-tlwg-loma-ttf fonts-tlwg-mono fonts-tlwg-mono-ttf fonts-tlwg-norasi fonts-tlwg-norasi-ttf fonts-tlwg-purisa
  fonts-tlwg-purisa-ttf fonts-tlwg-sawasdee fonts-tlwg-sawasdee-ttf fonts-tlwg-typewriter fonts-tlwg-typewriter-ttf
  fonts-tlwg-typist fonts-tlwg-typist-ttf fonts-tlwg-typo fonts-tlwg-typo-ttf fonts-tlwg-umpush fonts-tlwg-umpush-ttf
  fonts-tlwg-waree fonts-tlwg-waree-ttf g++-11 gcc-12-base gedit gedit-common genisoimage gir1.2-amtk-5 gir1.2-gck-1
  gir1.2-gcr-3 gir1.2-goa-1.0 gir1.2-gtksource-300 gir1.2-gtksource-4 gir1.2-gweather-3.0 gir1.2-javascriptcoregtk-4.0
  gir1.2-json-1.0 gir1.2-nma-1.0 gir1.2-snapd-1 gir1.2-soup-2.4 gir1.2-tepl-6 gir1.2-webkit2-4.0 gnome-bluetooth
  gnome-bluetooth-common gnome-mahjongg gnome-mines gnome-sudoku gnome-todo gnome-video-effects
  gstreamer1.0-clutter-3.0 guile-2.2-libs guile-3.0-libs irqbalance libabsl20210324 libappstream4 libatk1.0-data
  libblockdev-crypto2 libblockdev-fs2 libblockdev-loop2 libblockdev-part-err2 libblockdev-part2 libblockdev-swap2
  libblockdev-utils2 libblockdev2 libboost-filesystem1.74.0 libboost-iostreams1.74.0 libboost-locale1.74.0
  libboost-regex1.74.0 libboost-thread1.74.0 libbpf0 libcamel-1.2-63 libcbor0.8 libcephfs2 libcheese-gtk25 libcheese8
  libclutter-1.0-0 libclutter-1.0-common libclutter-gst-3.0-0 libclutter-gtk-1.0-0 libcogl-common libcogl-pango20
```

Version:

```
qjrlazo@workstation:~$ git --version
git version 2.43.0
qjrlazo@workstation:~$ S
```

Configure:

```
qjrlazo@workstation:~$ git config --global user.name "Jessie-Lazo"
qjrlazo@workstation:~$ git config --global user.email "jessielazo1441@gmail.com"
qjrlazo@workstation:~$ git config user.name\
> ^C
qjrlazo@workstation:~$ git config user.name
Jessie-Lazo
qjrlazo@workstation:~$ git config user.email
jessielazo1441@gmail.com
Show Apps rkstation:~$
```

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*

```
qjrlazo@workstation:~/CPE232_jessielazo$ which git
/usr/bin/git
qjrlazo@workstation:~/CPE232_jessielazo$
```

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

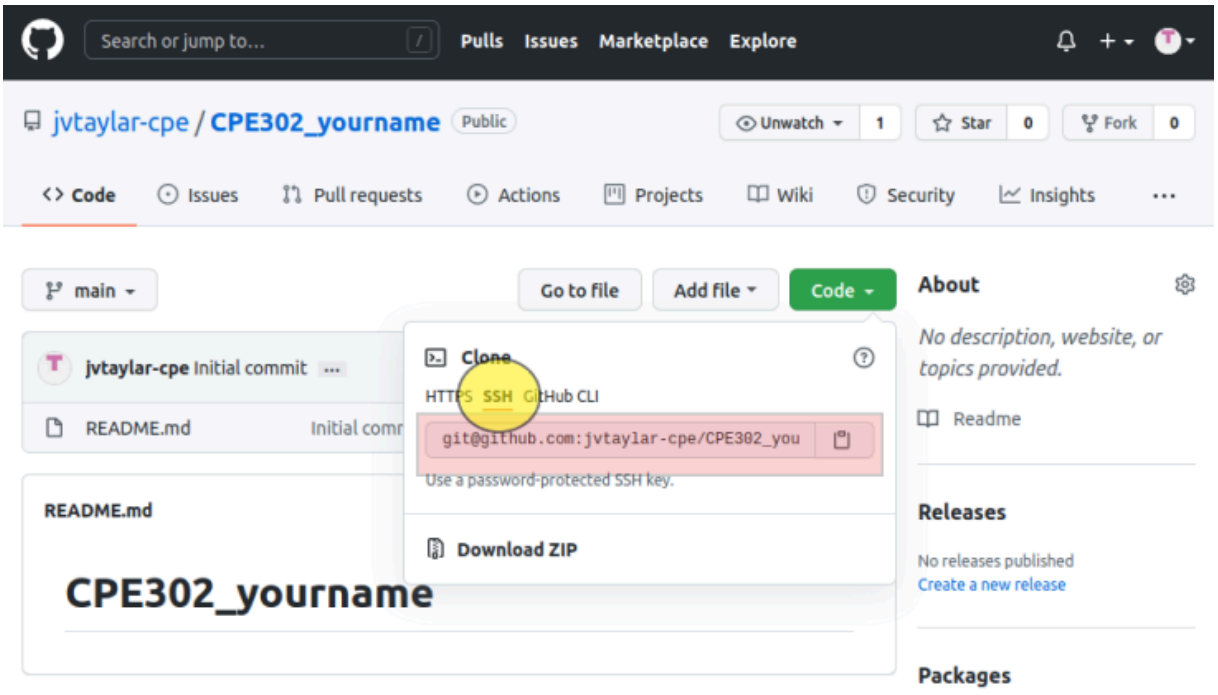
```
qjrlazo@workstation:~/CPE232_jessielazo$ which git
/usr/bin/git
qjrlazo@workstation:~/CPE232_jessielazo$
```

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

```
qjrlazo@workstation:~/CPE232_jessielazo$ git --version
git version 2.43.0
qjrlazo@workstation:~/CPE232_jessielazo$
```

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.
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.

```
qjrlazo@workstation:~$ ls
CPE232_jessielazo  del  Desktop
qjrlazo@workstation:~$
```

```
qjrlazo@workstation:~/CPE232_jessielazo$ ls
README.md
qjrlazo@workstation:~/CPE232_jessielazo$
```

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*

```
README.md
qjrlazo@workstation:~/CPE232_jessielazo$ cat ~/.gitconfig
[user]
        name = Jessie-Lazo
        email = jessielazo1441@gmail.com
qjrlazo@workstation:~/CPE232_jessielazo$
```

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.

```
qjrlazo@workstation: ~/CPE232_jessielazo
  GNU nano 7.2                    README.md
# CPE232_jessielazo
```

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?

   Changes not staged for commit

```
qjrlazo@workstation:~/CPE232_jessielazo$ 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")
qjrlazo@workstation:~/CPE232_jessielazo$
```

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

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.

```
no changes added to commit (use "git add" and/or "git commit -a")
qjrlazo@workstation:~/CPE232_jessielazo$ git add README.md
qjrlazo@workstation:~/CPE232_jessielazo$ git commit -m "first commit"
[main e2117e6] first commit
 1 file changed, 1 insertion(+), 1 deletion(-)
qjrlazo@workstation:~/CPE232_jessielazo$
```

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*.

```
qjrlazo@workstation:~/CPE232_jessielazo$ git push origin main
Username for 'https://github.com': jessie-lazo
Password for 'https://jessie-lazo@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/about-remote-repositories#cloning-with-h
ttps-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/Jessie-Lazo/CPE232_jessielazo.git/'
qjrlazo@workstation:~/CPE232_jessielazo$
```

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: