Name: Maravilla, Keith Dominic T.	**Date Performed: August 25, 2022**
Course/Section: CPE232 - CPE31S23	**Date Submitted: August 25, 2022**
Instructor: Engr. Jonathan 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.

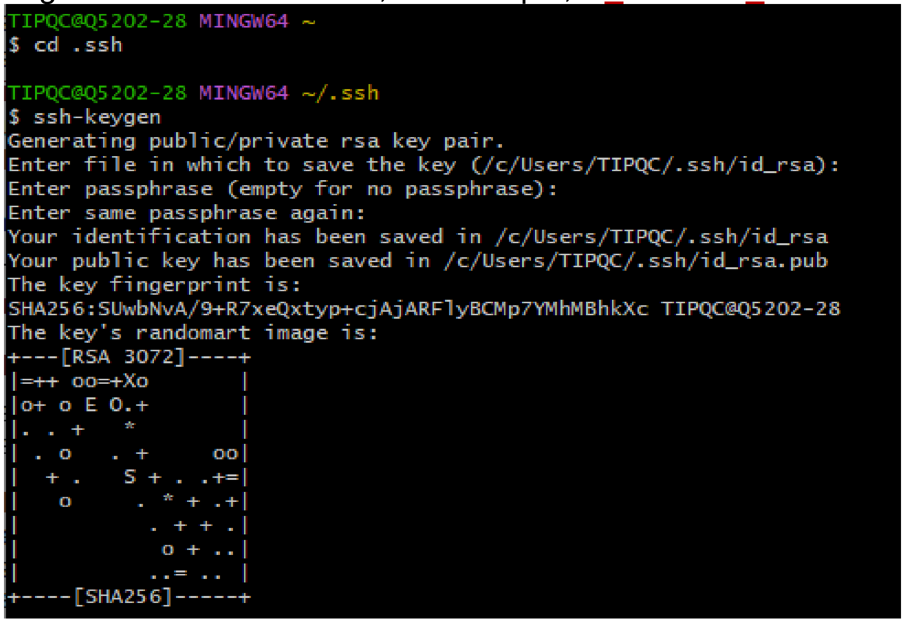
SSH introduced public key authentication as a more secure alternative to the older .rhosts authentication. It improved security by avoiding the need to have password stored in files and eliminated the possibility of a compromised server stealing the user's password.

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

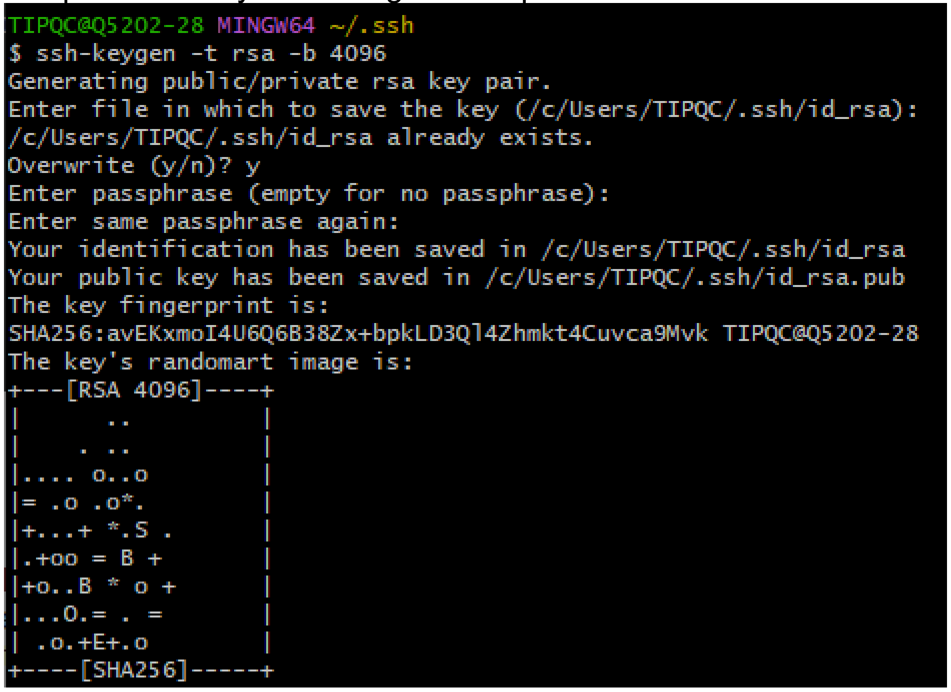
1. The simplest way to generate a key pair is to run *ssh-keygen* without

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*.



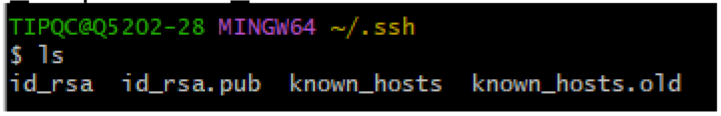
```
TIPQC@Q5202-28 MINGW64 ~
$ cd .ssh

TIPQC@Q5202-28 MINGW64 ~/.ssh
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/TIPQC/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/TIPQC/.ssh/id_rsa
Your public key has been saved in /c/Users/TIPQC/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:SUwbNvA/9+R7xeQxtyp+cjAjARFlyBCMp7YMhMBhkXc TIPQC@Q5202-28
The key's randomart image is:
+---[RSA 3072]----+
|=++ oo=+Xo       |
|o+ o E O.+       |
|. . +   *        |
| . o   . +     oo|
|  + .   S + . .+=|
|   o     . * + .+|
|          . + + .|
|           o + ..|
|          ..= .. |
+----[SHA256]-----+
```

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.



```
TIPQC@Q5202-28 MINGW64 ~/.ssh
$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/TIPQC/.ssh/id_rsa):
/c/Users/TIPQC/.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 /c/Users/TIPQC/.ssh/id_rsa
Your public key has been saved in /c/Users/TIPQC/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:avEKxmoI4U6Q6B38Zx+bpkLD3Ql4Zhmkt4Cuvca9Mvk TIPQC@Q5202-28
The key's randomart image is:
+---[RSA 4096]----+
|      ..         |
|    . ..         |
|.... o..o        |
|= .o .o*.        |
|+...+ *.S .      |
|.+oo = B +       |
|+o..B * o +      |
|...O.= . =       |
| .o.+E+.o        |
+----[SHA256]-----+
```

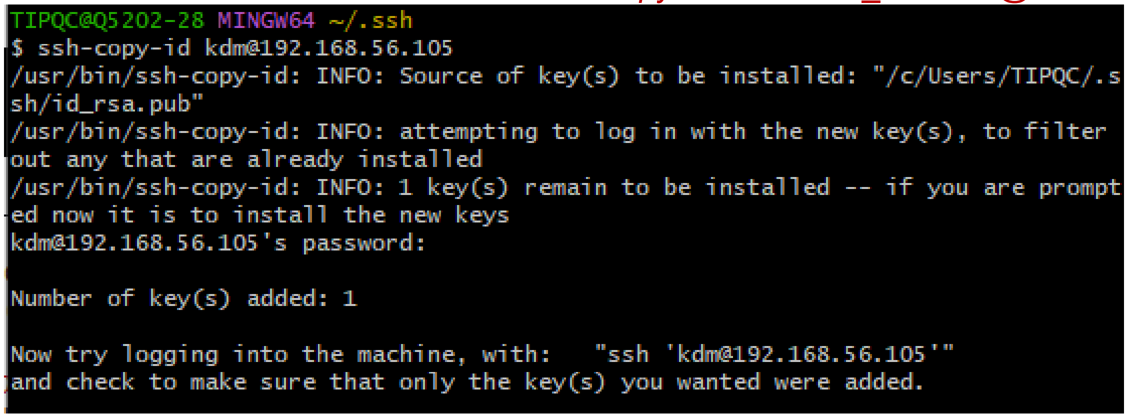
3. When asked for a passphrase, just press enter. The passphrase is used for encrypting the key, so that it cannot be used even if someone obtains the private key file. The passphrase should be cryptographically strong.
4. Verify that you have created the key by issuing the command *ls -la .ssh.* The command should show the .ssh directory containing a pair of keys. For example, id_rsa.pub and id_rsa.



```
TIPQC@Q5202-28 MINGW64 ~/.ssh
$ ls
id_rsa   id_rsa.pub   known_hosts   known_hosts.old
```

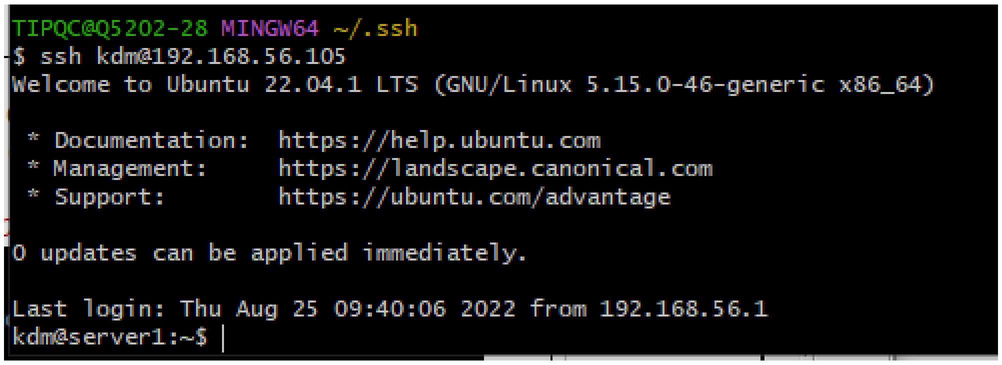
Task 2: Copying the Public Key to the remote servers

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*

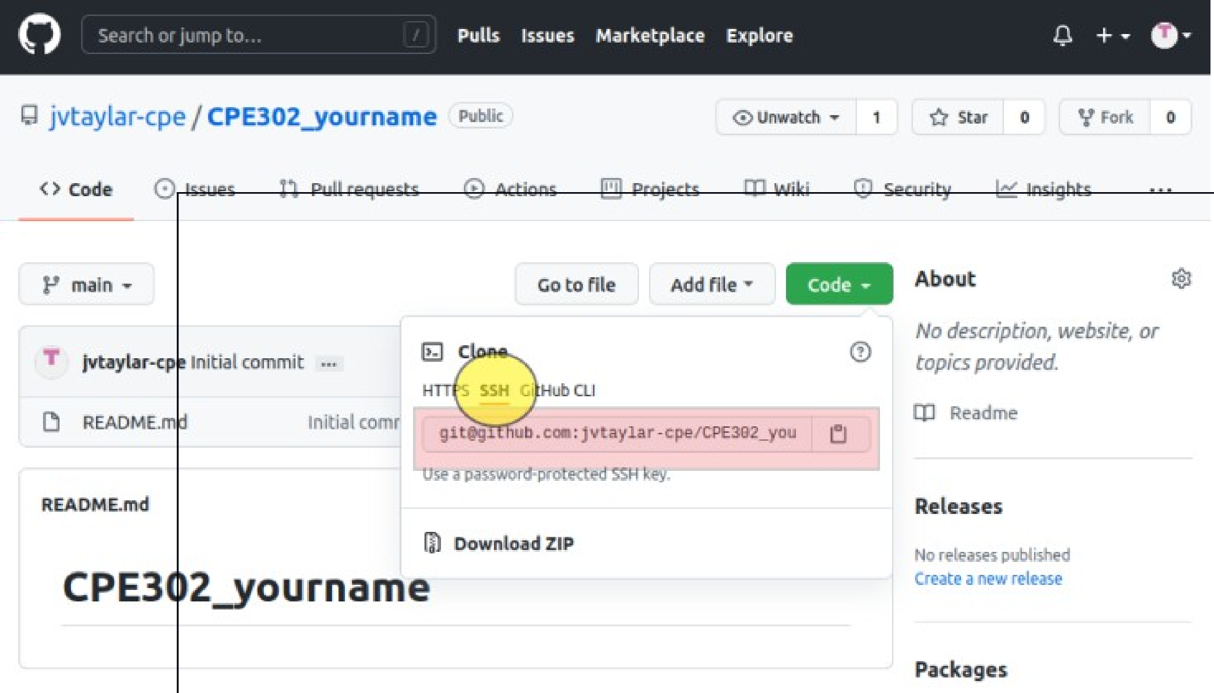


```
TIPQC@Q5202-28 MINGW64 ~/.ssh
$ ssh-copy-id kdm@192.168.56.105
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/c/Users/TIPQC/.s
sh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompt
ed now it is to install the new keys
kdm@192.168.56.105's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'kdm@192.168.56.105'"
and check to make sure that only the key(s) you wanted were added.
```

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?
   FOR SERVER 1:

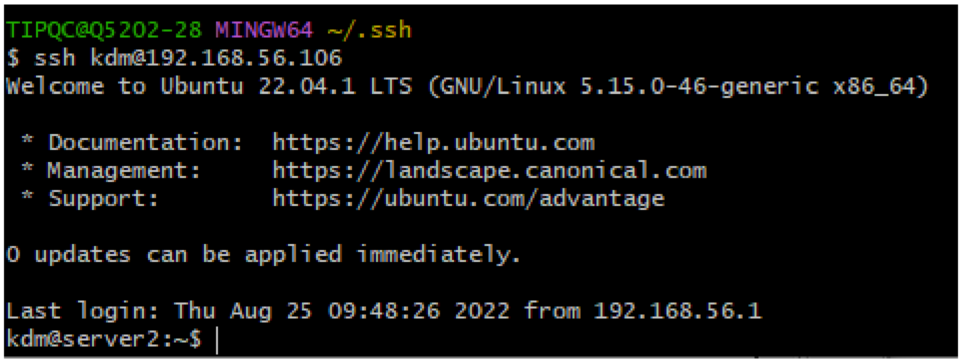


```
TIPQC@Q5202-28 MINGW64 ~/.ssh
$ ssh kdm@192.168.56.105
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-46-generic x86_64)

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

0 updates can be applied immediately.

Last login: Thu Aug 25 09:40:06 2022 from 192.168.56.1
kdm@server1:~$ |
```



FOR

SERVER 2:



```
TIPQC@Q5202-28 MINGW64 ~/.ssh
$ ssh kdm@192.168.56.106
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.

Last login: Thu Aug 25 09:48:26 2022 from 192.168.56.1
kdm@server2:~$ 
```

Observation: I noticed that once I copied the public key authentication, it states that I can try to use the ssh command. Upon executing the command, it can immediately access the SSH of a certain server. It did not ask for a password because the public key is copied to each server.

Reflections:

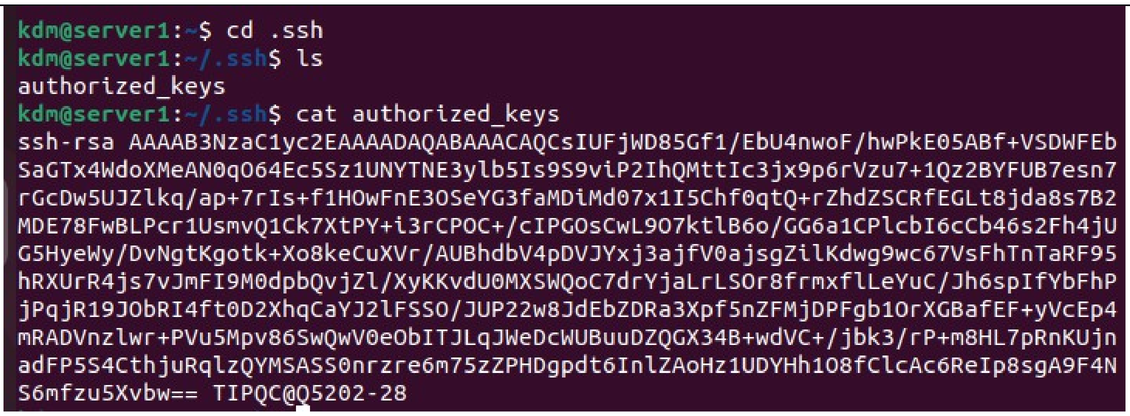
Answer the following:

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

   SSH program is higly secured when properly managed or configured. It allows users to remotely control and manage remote servers. Another thing is that SSH program uses a client-server paradigm or method which allows clients and servers to communicate securely.

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

   We can identify if there's already a public key installed to the remote servers if there's a file authorized_keys inside the .ssh directory in a certain server.
   Like this one for example:



```
kdm@server1:~$ cd .ssh
kdm@server1:~/.ssh$ ls
authorized_keys
kdm@server1:~/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQCsIUFjWD85Gf1/EbU4nwoF/hwPkE05ABf+VSDWFEb
SaGTx4WdoXMeAN0qO64Ec5Sz1UNYTNE3ylb5Is9S9viP2IhQMttIc3jx9p6rVzu7+1Qz2BYFUB7esn7
rGcDw5UJZlkq/ap+7rIs+f1HOwFnE3OSeYG3faMDiMd07x1I5Chf0qtQ+rZhdZSCRfEGLt8jda8s7B2
MDE78FwBLPcr1UsmvQ1Ck7XtPY+i3rCPOC+/cIPGOsCwL9O7ktlB6o/GG6a1CPlcbI6cCb46s2Fh4jU
G5HyeWy/DvNgtKgotk+Xo8keCuXVr/AUBhdbV4pDVJYxj3ajfV0ajsgZilKdwg9wc67VsFhTnTaRF95
hRXUrR4js7vJmFI9M0dpbQvjZl/XyKKvdU0MXSWQoC7drYjaLrLSOr8frmxflLeYuC/Jh6spIfYbFhP
jPqjR19JObRI4ft0D2XhqCaYJ2lFSSO/JUP22w8JdEbZDRa3Xpf5nZFMjDPFgb1OrXGBafEF+yVcEp4
mRADVnzlwr+PVu5Mpv86SwQwV0eObITJLqJWeDcWUBuuDZQGX34B+wdVC+/jbk3/rP+m8HL7pRnKUjn
adFP5S4CthjuRqlzQYMSASS0nrzre6m75zZPHDgpdt6InlZAoHz1UDYHh1O8fClcAc6ReIp8sgA9F4N
S6mfzu5Xvbw== TIPQC@Q5202-28
```

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:

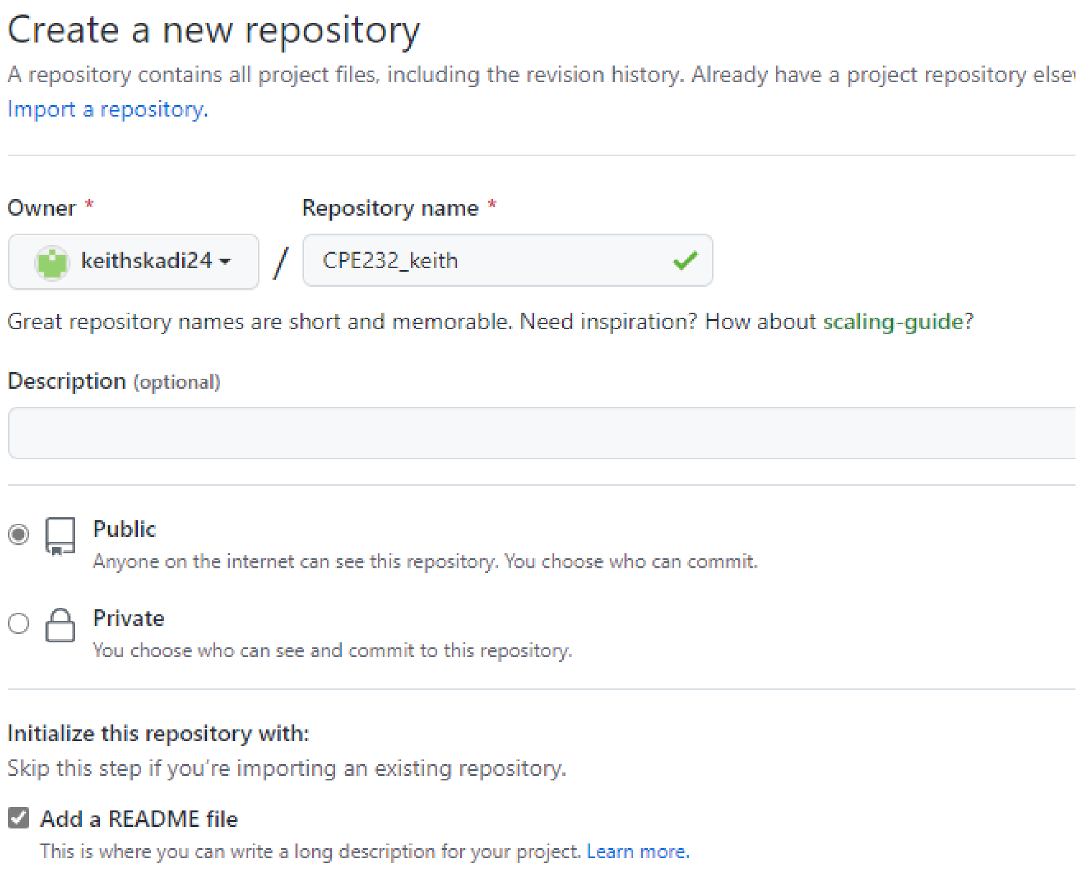
- Creating a repository
- Forking a repository
- Managing files
- Being social

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



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository else

Import a repository.

Owner * keithskadi24 / Repository name * CPE232_keith

Great repository names are short and memorable. Need inspiration? How about scaling-guide?

Description (optional)

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

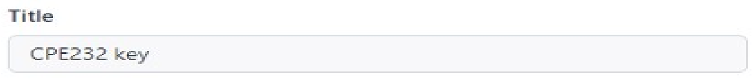
Private You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

[✓ Add a README file] This is where you can write a long description for your project. Learn more.

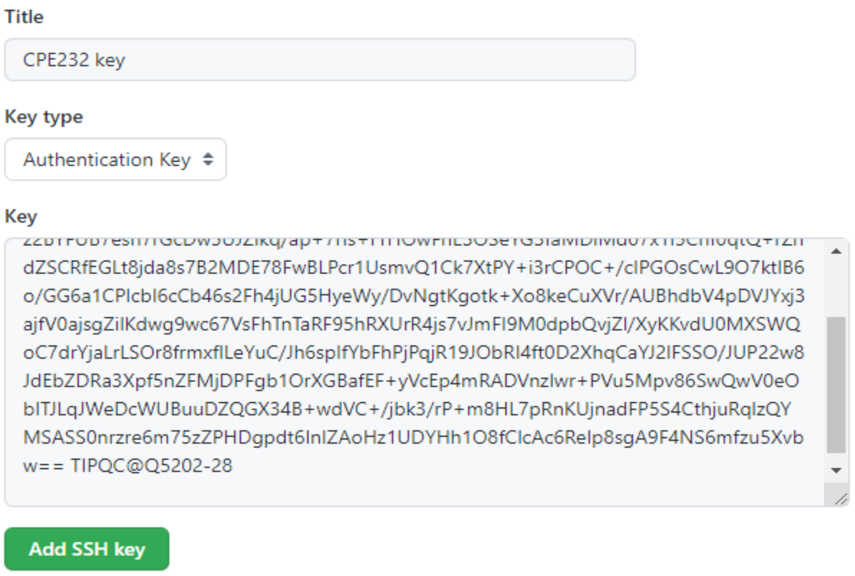
b. Create a new SSH key on GitHub. Go your profile's setting and click SSH and GPG keys. If there is an existing key, make sure to delete it. To create a new SSH keys, click New SSH Key. Write CPE232 key as the title of the key.



Title

CPE232 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.



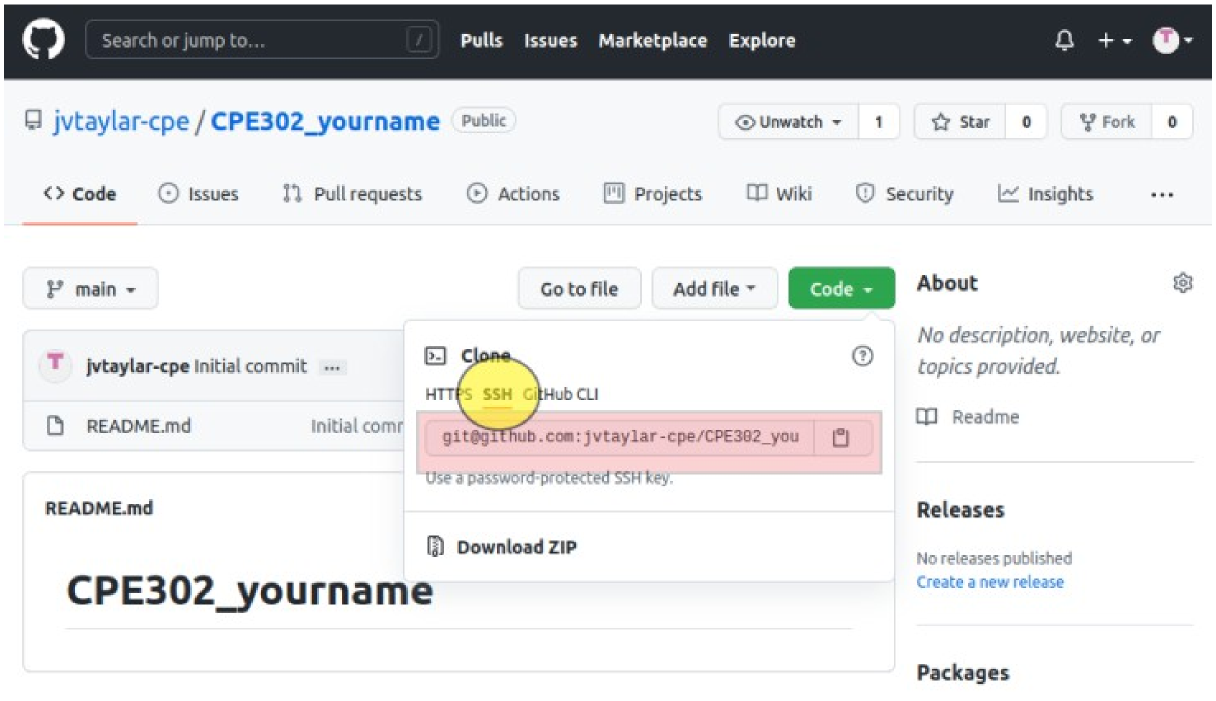
Title: CPE232 key

Key type: Authentication Key

Key:

```
dZSCRfEGLt8jda8s7B2MDE78FwBLPcr1UsmvQ1Ck7XtPY+i3rCPOC+/cIPGOsCwL9O7ktIB6
o/GG6a1CPlcbl6cCb46s2Fh4jUG5HyeWy/DvNgtKgotk+Xo8keCuXVr/AUBhdbV4pDVJYxj3
ajfV0ajsgZiIKdwg9wc67VsFhTnTaRF95hRXUrR4js7vJmFI9M0dpbQvjZI/XyKKvdU0MXSWQ
oC7drYjaLrLSOr8frmxflLeYuC/Jh6splfYbFhPjPqjR19JObRI4ft0D2XhqCaYJ2IFSSO/JUP22w8
JdEbZDRa3Xpf5nZFMjDPFgb1OrXGBafEF+yVcEp4mRADVnzlwr+PVu5Mpv86SwQwV0eO
bITJLqJWeDcWUBuuDZQGX34B+wdVC+/jbk3/rP+m8HL7pRnKUjnadFP5S4CthjuRqlzQY
MSASS0nrzre6m75zZPHDgpdt6lnIZAoHz1UDYHh1O8fClcAc6Relp8sgA9F4NS6mfzu5Xvb
w== TIPQC@Q5202-28
```

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.



jvtaylar-cpe Initial commit

README.md Initial comm

README.md

CPE302_yourname

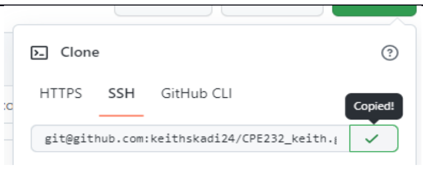
git@github.com:jvtaylar-cpe/CPE302_you

Use a password-protected SSH key.

No description, website, or topics provided.

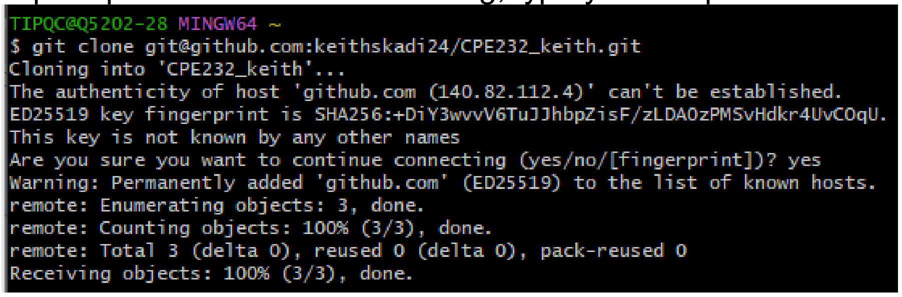
No releases published

Create a new release



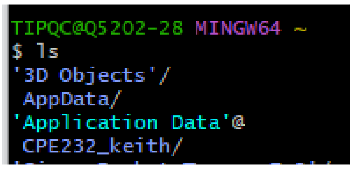
```
git@github.com:keithskadi24/CPE232_keith.g
```

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.

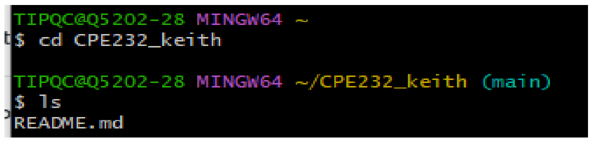


```
TIPQC@Q5202-28 MINGW64 ~
$ git clone git@github.com:keithskadi24/CPE232_keith.git
Cloning into 'CPE232_keith'...
The authenticity of host 'github.com (140.82.112.4)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other 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.
```

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.



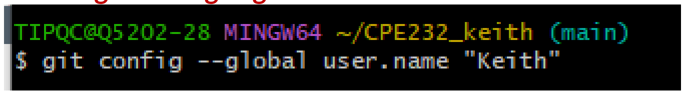
```
TIPQC@Q5202-28 MINGW64 ~
$ ls
'3D Objects'/
 AppData/
'Application Data'@
 CPE232_keith/
```



```
TIPQC@Q5202-28 MINGW64 ~
$ cd CPE232_keith

TIPQC@Q5202-28 MINGW64 ~/CPE232_keith (main)
$ ls
README.md
```

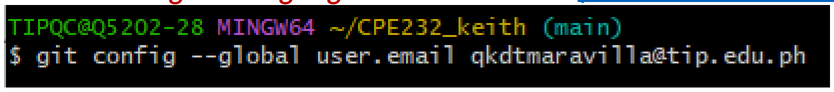
g. Use the following commands to personalize your git.

- *git config --global user.name "Your Name"*



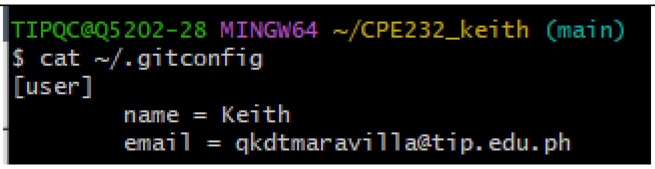
```
TIPQC@Q5202-28 MINGW64 ~/CPE232_keith (main)
$ git config --global user.name "Keith"
```

- *git config --global user.email* yourname@email.com



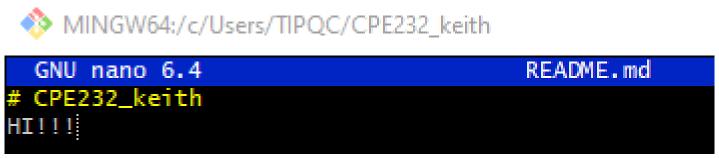
```
TIPQC@Q5202-28 MINGW64 ~/CPE232_keith (main)
$ git config --global user.email qkdtmaravilla@tip.edu.ph
```

- Verify that you have personalized the config file using the command *cat ~/.gitconfig*



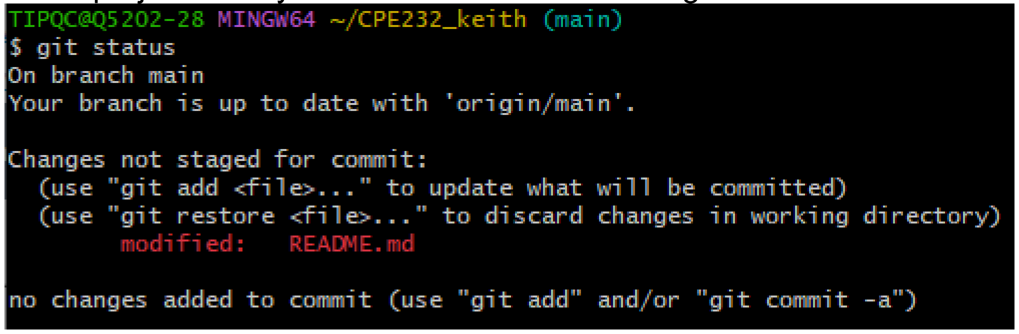
```
TIPQC@Q5202-28 MINGW64 ~/CPE232_keith (main)
$ cat ~/.gitconfig
[user]
        name = Keith
        email = qkdtmaravilla@tip.edu.ph
```

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.



```
MINGW64:/c/Users/TIPQC/CPE232_keith
  GNU nano 6.4                        README.md
# CPE232_keith
HI!!!
```

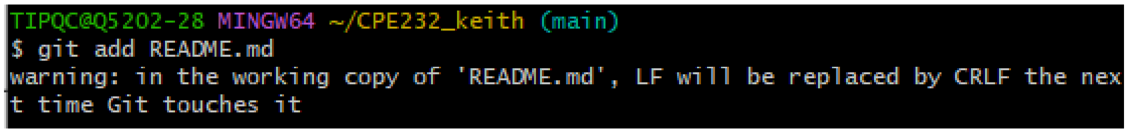
i. Use the *git status* command to display the state of the working directory and the staging area. This command shows which changes have been staged, which haven't, and which files aren't being tracked by Git. Status output does not show any information regarding the committed project history. What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



```
TIPQC@Q5202-28 MINGW64 ~/CPE232_keith (main)
$ 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")
```

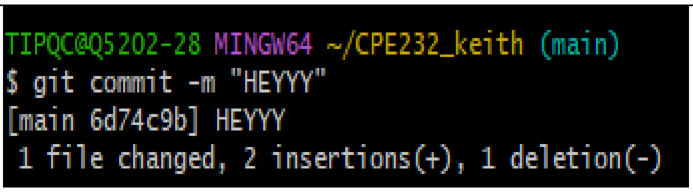
It shows the status of the repository, including the files that we want to add, restore, and commit.

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



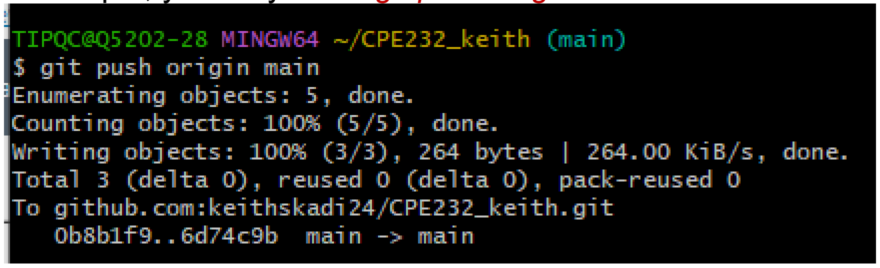
```
TIPQC@Q5202-28 MINGW64 ~/CPE232_keith (main)
$ git add README.md
warning: in the working copy of 'README.md', LF will be replaced by CRLF the nex
t time Git touches it
```

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.



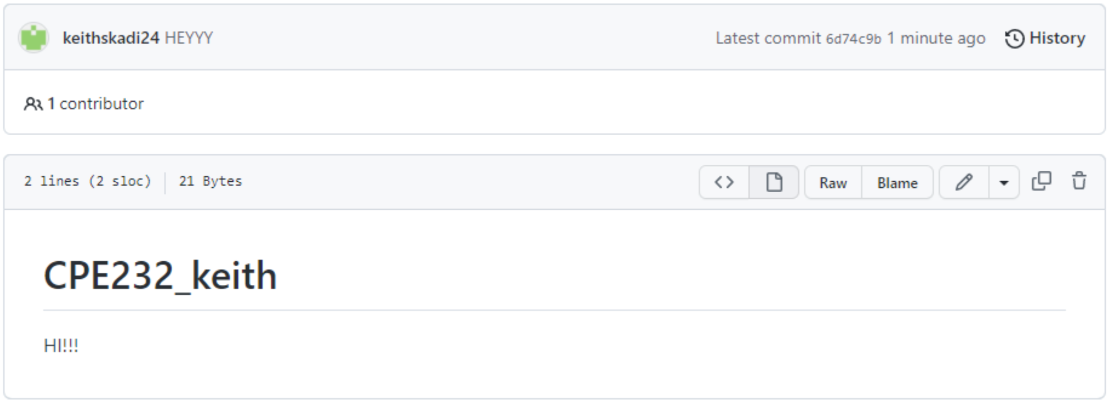
```
TIPQC@Q5202-28 MINGW64 ~/CPE232_keith (main)
$ git commit -m "HEYYY"
[main 6d74c9b] HEYYY
 1 file changed, 2 insertions(+), 1 deletion(-)
```

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 push origin main.*



```
TIPQC@Q5202-28 MINGW64 ~/CPE232_keith (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 264 bytes | 264.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:keithskadi24/CPE232_keith.git
   0b8b1f9..6d74c9b  main -> main
```

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.



keithskadi24 HEYYY   Latest commit 6d74c9b 1 minute ago   History

1 contributor

2 lines (2 sloc) | 21 Bytes

CPE232_keith

HI!!!

I observed the time it was last commited, also the contents of the README.md file which I changed awhile ago. And also there is the message shown when I executed the git commit command.

Reflections:
Answer the following:
3. What sort of things have we so far done to the remote servers using ansible commands?
   Using these commands, we can connect local machine and remote servers to be manageable or controllable in a certain unit. Using ansible commands, we

can set up and maintain remote servers from a central point or location. Another thing is that we can manage several remote servers. We can also create or initialize keys to authenticate our remote servers.

4. How important is the inventory file?

   Inventory file is important to reduce the interpretation time as we define and add variables to the inventory file and obviously reduces the confusion as every log will be monitored. It is also important to define the host of a system we are working on. Another thing is that update and changes will be much more easier.

Conclusions/Learnings:

In this activity, I learned about creating SSH keys to authenticate login in a certain server. I also learned the importance of having keys in managing servers. I conclude that, having the knowledge to manage private and public keys are essential as it can reduce the time to access servers and also to authenticate our work. I also learned in this activity how to manage my Github account like adding repositories and adding keys to it. Overall, this activity helped me gained knowledge about keys in managing servers and about managing Github.

I affirm that I will not give or receive any unauthorized help on this activity/exam and that all work will be my own.
-Keith Maravilla