

<b>Name: Jozette Fermin</b>	<b>Date Performed: August 22, 2023</b>
<b>Course/Section: CPE31S4</b>	<b>Date Submitted: August 29, 2023</b>
<b>Instructor: Engr. Jonathan Taylar</b>	<b>Semester and SY: 1st sem 2023-2024</b>
<b>Activity 2: SSH Key-Based Authentication and Setting up Git</b>	
<b>1. Objectives:</b> 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	
<b>Part 1: Discussion</b>  It is assumed that you are already done with the last Activity ( <b>Activity 1: Configure Network using Virtual Machines</b> ). <i>Provide screenshots for each task.</i>  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.	
<b>What is ssh-keygen?</b>  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.	
<b>SSH Keys and Public Key Authentication</b>  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.	
<b>Task 1: Create an SSH Key Pair for User Authentication</b> 1. The simplest way to generate a key pair is to run <i>ssh-keygen</i> 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`.

```
jozette@ManageNode:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jozette/.ssh/id_rsa):
/home/jozette/.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/jozette/.ssh/id_rsa.
Your public key has been saved in /home/jozette/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:V49hgNzsmZaN3tugqDytc8KsZqD/GdcISpLs8ZZijF4 jozette@ManageNode
The key's randomart image is:
+---[RSA 2048]-----+
|      . +.      |
|      o o.      |
|      . *+      |
|..      Bo.+    |
|oo. .   So... . |
|+o+... o .. o   |
|.*.E.oo... . +  |
|+ + o=* + . . . |
| o.++.+B        |
+---[SHA256]-----+
jozette@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.
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`.

```

jozette@ManageNode:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jozette/.ssh/id_rsa):
/home/jozette/.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/jozette/.ssh/id_rsa.
Your public key has been saved in /home/jozette/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:OE0p+MBQcAtQLQ2yeRrgZQq7iMMzhC/DGrHeQxFvod4 jozette@ManageNode
The key's randomart image is:
+---[RSA 4096]-----+
|o*Xo.                |
|====*o.  .           |
|B+.+=o. o            |
|*O. +o =             |
|@=.o E+ S            |
|o==                  |
|.. o                 |
|                      |
+-----[SHA256]-----+
jozette@ManageNode:~$ clear

jozette@ManageNode:~$ ls -la .ssh
total 20
drwx----- 2 jozette jozette 4096 Aug 22 16:58 .
drwxr-xr-x 18 jozette jozette 4096 Aug 22 17:38 ..
-rw----- 1 jozette jozette 3243 Aug 22 17:44 id_rsa
-rw-r--r-- 1 jozette jozette 744 Aug 22 17:44 id_rsa.pub
-rw-r--r-- 1 jozette jozette 888 Aug 15 18:08 known_hosts

```

## 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? Yes, the connection asks for a password because it's the first time connecting to the server and it needs authentication.

```
jozette@ManageNode:~$ ssh-copy-id -i ~/.ssh/id_rsa jozette@controlNode2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/jozette/.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 prompted now it is to install the new keys
jozette@controlNode2's password:
```

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'jozette@controlNode2'"  
and check to make sure that only the key(s) you wanted were added.

```
jozette@ManageNode:~$ ssh-copy-id -i ~/.ssh/id_rsa jozette@controlNode1
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/jozette/.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 prompted now it is to install the new keys
jozette@controlNode1's password:
```

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'jozette@controlNode1'"  
and check to make sure that only the key(s) you wanted were added.

## Reflections:

Answer the following:

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

SSH is like a secret tunnel between your computer and another computer, like a server. This tunnel keeps the information you send and receive safe from anyone trying to eavesdrop. It's like having a private conversation even if you're talking on a crowded street.

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

To check if you've installed your public key on a remote server, you can attempt to connect to the server using SSH. If you're able to log in without being prompted for a password, then your public key is installed and working correctly. If you're prompted for a password, then your public key might not be set up on that server, or there could be a configuration issue.

## 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

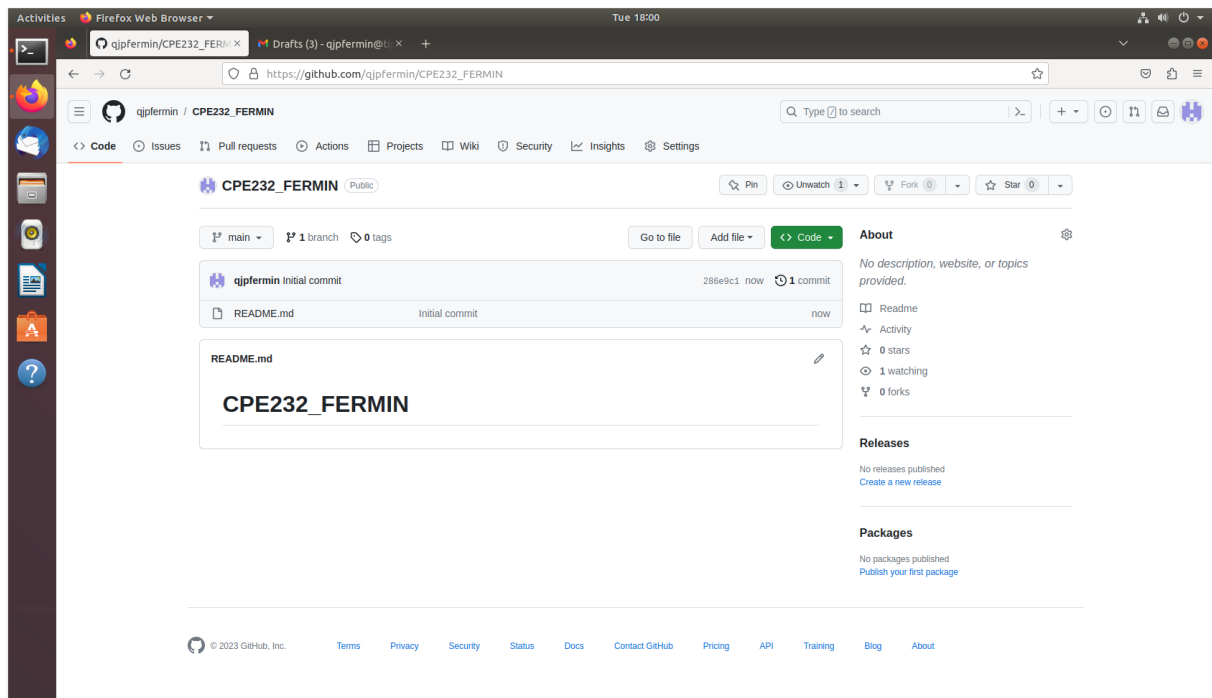
- Forking a repository
- Managing files
- Being social

### 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.

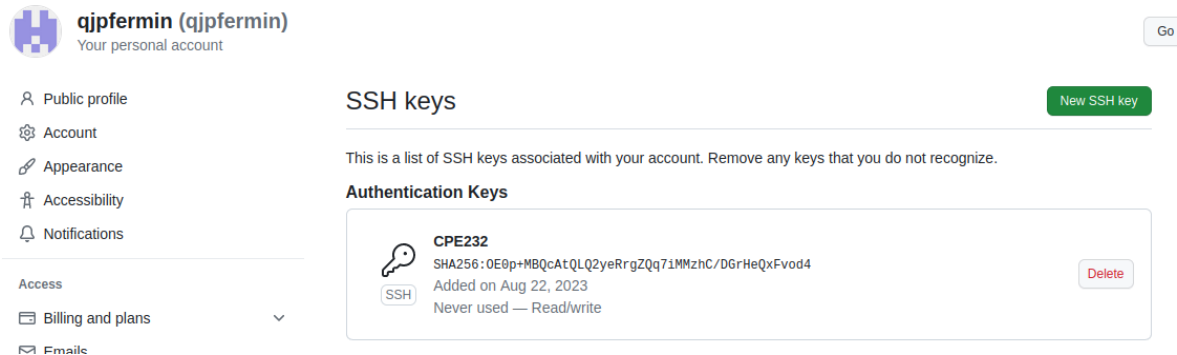
```
jozette@ManageNode:~$ which git
/usr/bin/git
jozette@ManageNode:~$ git --version
git version 2.17.1
jozette@ManageNode:~$
```

4. Using the browser in the local machine, go to [www.github.com](https://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

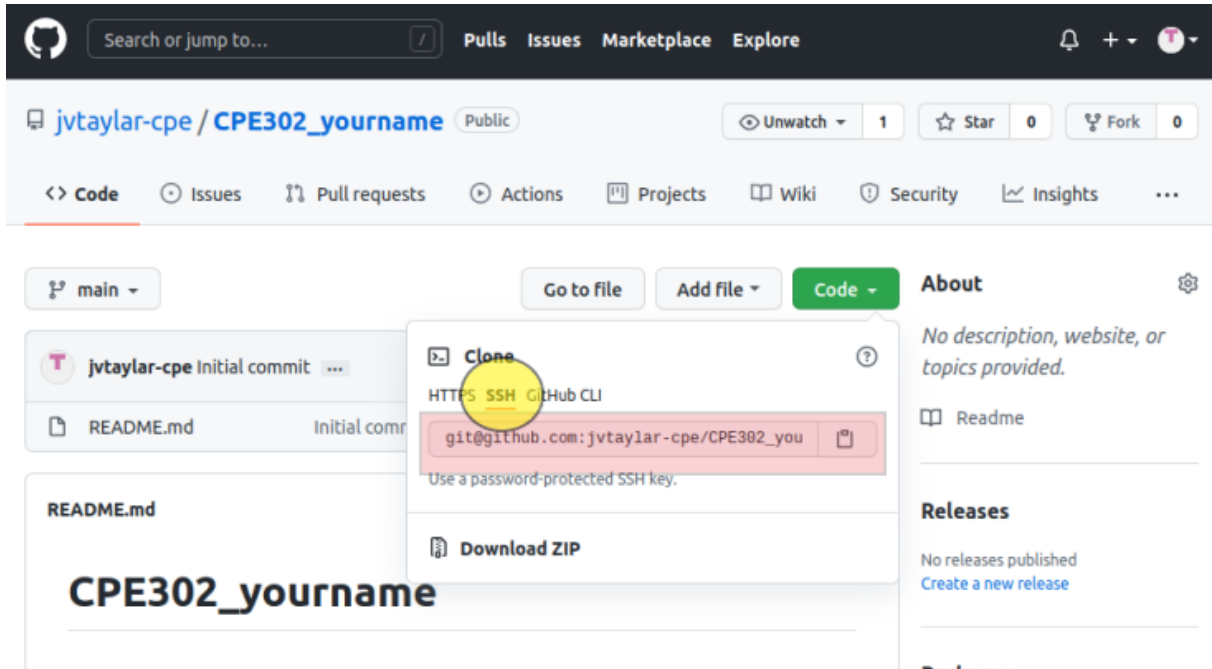
create a new SSH keys, click New SSH Key. Write CPE232 key as the title of the key.



- 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.

```
jozette@ManageNode:~$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCA4tepc7HtEaWUhp0n9hY3snXsQ0prBMULJ0t2C8cvWJ90Ze3rhbg+6mVvnf5k8CcW/VTmv2j103B2CraVtTp4NVPbPI5NN44Nb5r65ZxhCcpVJE58pFV2bbwyWMD
t1XZQ7L7+anQ1HU24YX08Lx41xssVkd+hJudd7r23bey+zLDsaue5GwFLTuplq460RVqYvcdRxsqJ3okhCRdrhuWj3N563ge5hfiWnt8nIOV+E998yzruc24lueNyOTL1V2qga5QhY004o5GlgVt4MfbH+LuynetGR1NRxZM
BvllzCly0Dz6KSnteRlyQ27NPW9g6ApM4wUwFQI2wQ54qc34e1/RmHO12FulQ85Zy5zxVbuQxR9aoS/cgskrLzxSTNNU41pFk5IGl+IsSAUT118tc1WGDsRdesyfMKJa1bvpVvyMKF0FjlztePHU0Jgcyte7FZZzXb+9e
WbI8lqhIw+FC2VVC7m1jsZP14T4Z28AB56DUKb8uB00TlMKqslg7YBhZWCk7Dn/MfP54SwdDYfsek3N357gALW60WfBBWUdzo4LGKK86ng0j3y4d1wQ38TVelc8lCTlerrRyqqnLcBTvIVktouviHlwQw0LmqVqt+8
0J5g5w5JA003+RSLTKlQhQDy8qEtNgbXRz0KLTeywyMHE7lLZ0== jozette@ManageNode
```

- 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:qjpfermin/CPE232\_yourname.git`. When prompted to continue connecting, type yes and press enter.

```
jozette@ManageNode:~$ git clone https://github.com/qjpfermin/CPE232_FERMIN.git
Cloning into 'CPE232_FERMIN'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
jozette@ManageNode:~$
```

- 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`.

```
jozette@ManageNode:~$ ls
CPE232_FERMIN Desktop Documents Downloads examples.desktop Music Pictures Public sample
jozette@ManageNode:~$ cd CPE232_FERMIN
jozette@ManageNode:~/CPE232_FERMIN$ ls
README.md
jozette@ManageNode:~/CPE232_FERMIN$
```

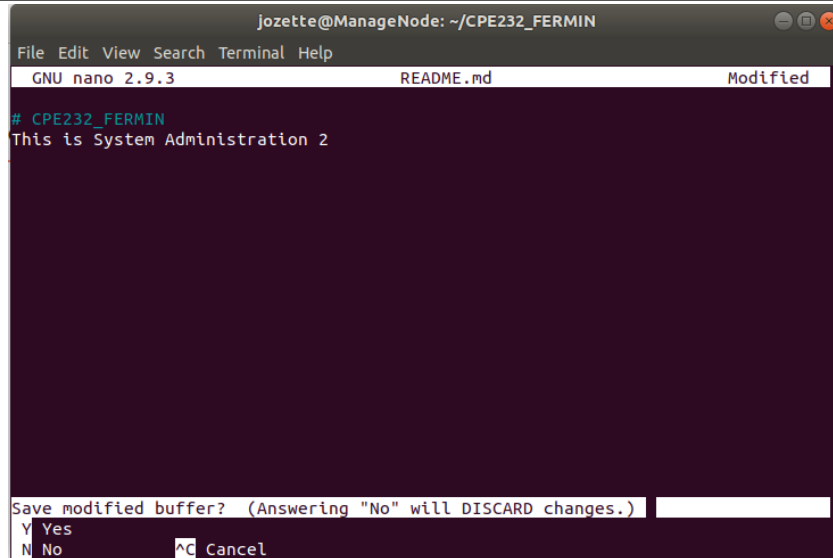
- g. Use the following commands to personalize your git.

- `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`

```
jozette@ManageNode:~/CPE232_FERMIN$ git config --global user.name "Jozette"
jozette@ManageNode:~/CPE232_FERMIN$ git config --global user.email qjpfermin@tip.edu.ph
jozette@ManageNode:~/CPE232_FERMIN$ cat ~/.gitconfig
[user]
  email = qjpfermin@tip.edu.ph
  name = Jozette
jozette@ManageNode:~/CPE232_FERMIN$
```

- 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.



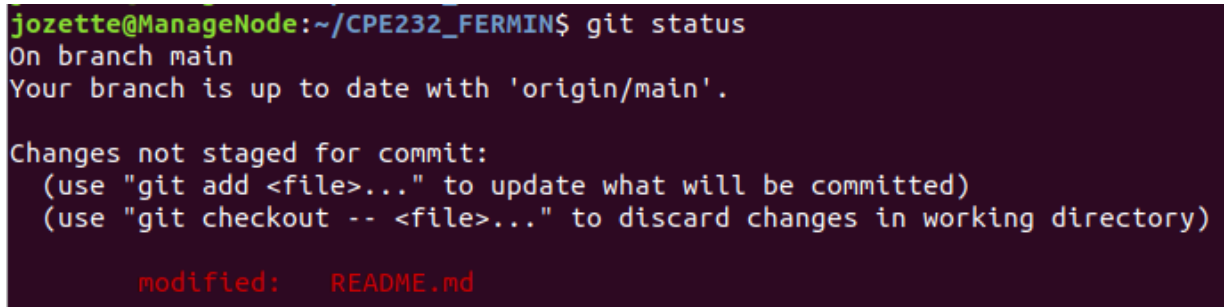


```
jozette@ManageNode: ~/CPE232_FERMIN
File Edit View Search Terminal Help
GNU nano 2.9.3 README.md Modified

# CPE232_FERMIN
This is System Administration 2

Save modified buffer? (Answering "No" will DISCARD changes.)
Y Yes
N No ^C Cancel
```

- 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?

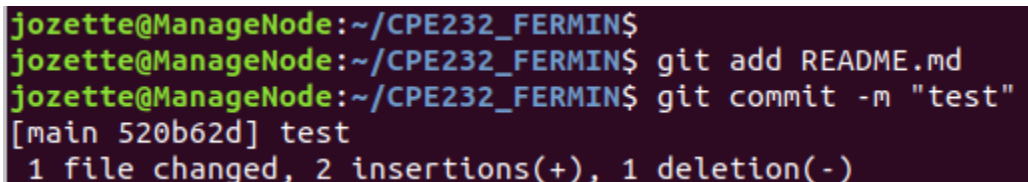


```
jozette@ManageNode:~/CPE232_FERMIN$ git status
On branch main
Your branch is up to date with 'origin/main'.

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
```

- 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.



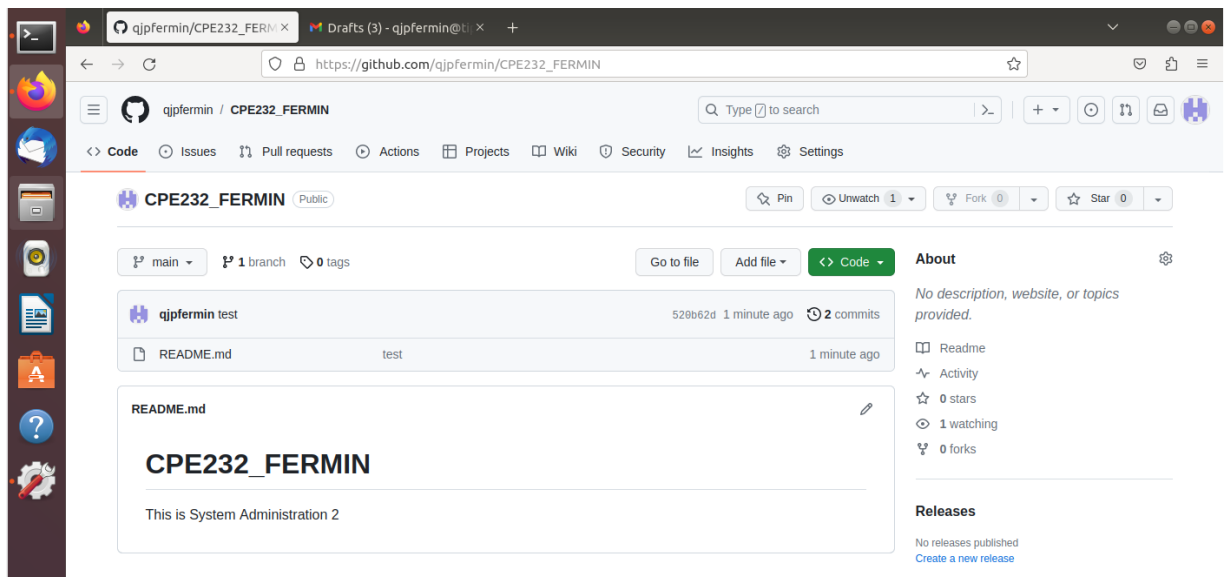
```
jozette@ManageNode:~/CPE232_FERMIN$
jozette@ManageNode:~/CPE232_FERMIN$ git add README.md
jozette@ManageNode:~/CPE232_FERMIN$ git commit -m "test"
[main 520b62d] test
1 file changed, 2 insertions(+), 1 deletion(-)
```

- 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*.



```
jozette@ManageNode:~/CPE232_FERMIN$ git push
Username for 'https://github.com': qjpfermin
Password for 'https://qjpfermin@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 284 bytes | 284.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/qjpfermin/CPE232_FERMIN.git
286e9c1..520b62d  main -> main
jozette@ManageNode:~/CPE232_FERMIN$
```

- 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?

Using Ansible commands, you can perform various tasks on remote servers, such as installing software, updating configurations, managing files, and even restarting services. Ansible helps you automate these tasks, making it easier to manage multiple servers simultaneously.

4. How important is the inventory file?

The inventory file is like a list of these remote computers. It's really important because it helps Ansible know which computers to talk to. Without this list, Ansible wouldn't know where to send its instructions. So, the inventory file is like a map that Ansible uses to find the right computers to work with.

### **Conclusions/Learnings:**

**In this hands-on activity, I've learned about two important concepts about SSH Key-Based Authentication and setting up Git.**

**SSH Key-Based Authentication is like having a special key to unlock the door to a remote server. It's more secure than using a password because it's really hard for someone else to replicate the key. I've discovered that when I install my public key on a remote server, I can connect to it without typing in a password every time. It's like my computer and the server have a secret handshake that lets me in.**

**Setting up Git is all about making it easier to work with code and collaborate with others. I've understood that Git is like a smart version control system that tracks changes to my code. It helps me keep track of different versions and lets me work on code with a team without causing conflicts. By using commands like "git add," "git commit," and "git push," I can manage my code and share it with others.**

**Overall, I've gained insights into these concepts that will make my remote server interactions more secure and my code management more efficient. SSH Key-Based Authentication safeguards my connections, and understanding Git helps me work on code projects smoothly. It's exciting to have these tools in my technical toolbox!**