Git Merges

1 hour 30 minutesFree

Rate Lab

**Introduction**

In this lab, you'll use your knowledge of Git and Git commit history to check out an existing repo and make some changes to it. You'll also test what you learned about rolling back commits after bad changes in order to fix a script in the repo and run it to produce the correct output.

What you'll do

* Check the status and history of an existing Git repo
* Create a branch
* Modify content on the branch
* Make rollback changes
* Merge the branch

You'll have 90 minutes to complete this lab.

Start the lab

You'll need to start the lab before you can access the materials in the virtual machine OS. To do this, click the green “Start Lab” button at the top of the screen.

**Note:** For this lab you are going to access the **Linux VM** through your **local SSH Client**, and not use the **Google Console** (**Open GCP Console** button is not available for this lab).

Start Lab

After you click the “Start Lab” button, you will see all the SSH connection details on the left-hand side of your screen. You should have a screen that looks like this:



**Accessing the virtual machine**

Please find one of the three relevant options below based on your device's operating system.

**Note:** Working with Qwiklabs may be similar to the work you'd perform as an **IT Support Specialist**; you'll be interfacing with a cutting-edge technology that requires multiple steps to access, and perhaps healthy doses of patience and persistence(!). You'll also be using **SSH** to enter the labs -- a critical skill in IT Support that you’ll be able to practice through the labs.

Option 1: Windows Users: Connecting to your VM

In this section, you will use the PuTTY Secure Shell (SSH) client and your VM’s External IP address to connect.

**Download your PPK key file**

You can download the VM’s private key file in the PuTTY-compatible **PPK** format from the Qwiklabs Start Lab page. Click on **Download PPK**.



**Connect to your VM using SSH and PuTTY**

1. You can download Putty from [here](https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe)
2. In the **Host Name (or IP address)** box, enter username@external\_ip\_address.

**Note:** Replace **username** and **external\_ip\_address** with values provided in the lab.



1. In the **Category** list, expand **SSH**.
2. Click **Auth** (don’t expand it).
3. In the **Private key file for authentication** box, browse to the PPK file that you downloaded and double-click it.
4. Click on the **Open** button.

**Note:** PPK file is to be imported into PuTTY tool using the Browse option available in it. It should not be opened directly but only to be used in PuTTY.



1. Click **Yes** when prompted to allow a first connection to this remote SSH server. Because you are using a key pair for authentication, you will not be prompted for a password.

**Common issues**

If PuTTY fails to connect to your Linux VM, verify that:

* You entered **<username>**@**<external ip address>** in PuTTY.
* You downloaded the fresh new PPK file for this lab from Qwiklabs.
* You are using the downloaded PPK file in PuTTY.

Option 2: OSX and Linux users: Connecting to your VM via SSH

**Download your VM’s private key file.**

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



**Connect to the VM using the local Terminal application**

A **terminal** is a program which provides a **text-based interface for typing commands**. Here you will use your terminal as an SSH client to connect with lab provided Linux VM.

1. Open the Terminal application.
   * To open the terminal in Linux use the shortcut key **Ctrl+Alt+t**.
   * To open terminal in **Mac** (OSX) enter **cmd + space** and search for **terminal**.
2. Enter the following commands.

**Note:** Substitute the **path/filename for the PEM** file you downloaded, **username** and **External IP Address**.

You will most likely find the PEM file in **Downloads**. If you have not changed the download settings of your system, then the path of the PEM key will be **~/Downloads/qwikLABS-XXXXX.pem**

chmod 600 ~/Downloads/qwikLABS-XXXXX.pem

ssh -i ~/Downloads/qwikLABS-XXXXX.pem username@External Ip Address



Option 3: Chrome OS users: Connecting to your VM via SSH

**Note:** Make sure you are not in **Incognito/Private mode** while launching the application.

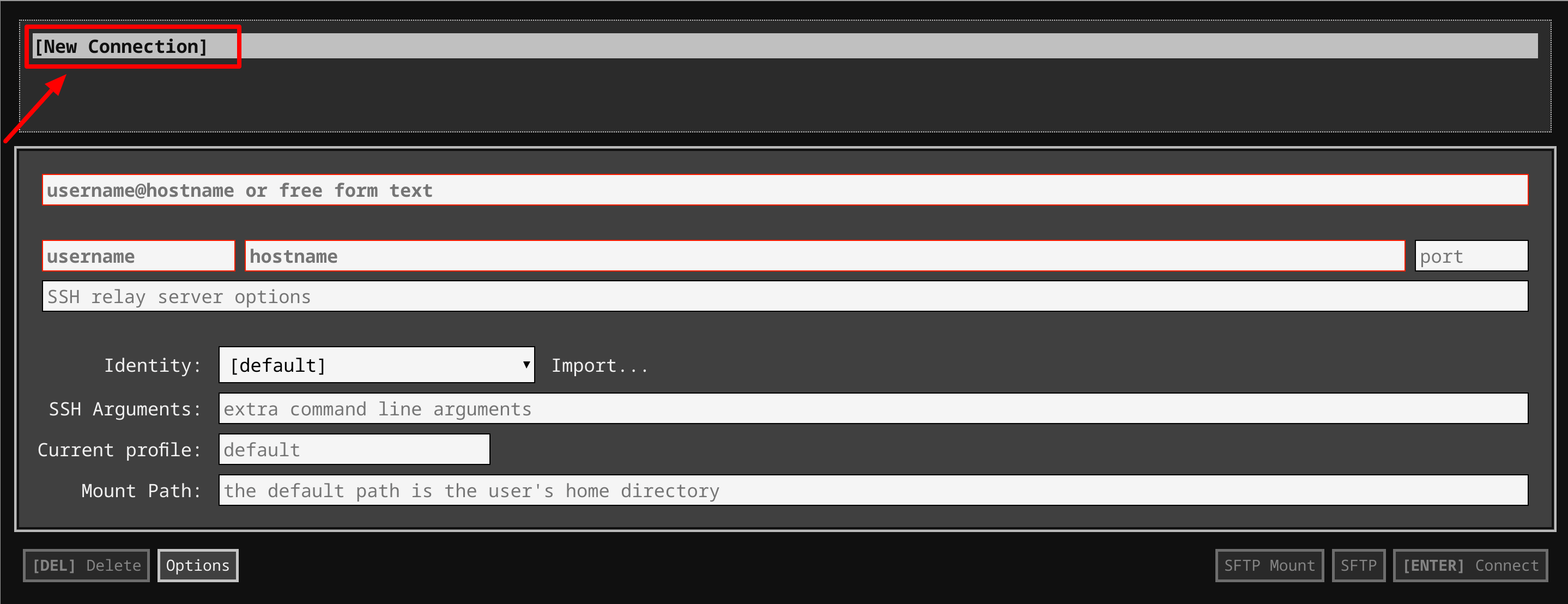
**Download your VM’s private key file.**

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



**Connect to your VM**

1. Add Secure Shell from [here](https://chrome.google.com/webstore/detail/secure-shell-app/pnhechapfaindjhompbnflcldabbghjo) to your Chrome browser.
2. Open the Secure Shell app and click on **[New Connection]**.



1. In the **username** section, enter the username given in the Connection Details Panel of the lab. And for the **hostname** section, enter the external IP of your VM instance that is mentioned in the Connection Details Panel of the lab.



1. In the **Identity** section, import the downloaded PEM key by clicking on the **Import…** button beside the field. Choose your PEM key and click on the **OPEN** button.

**Note:** If the key is still not available after importing it, refresh the application, and select it from the **Identity** drop-down menu.

1. Once your key is uploaded, click on the **[ENTER] Connect** button below.



1. For any prompts, type **yes** to continue.
2. You have now successfully connected to your Linux VM.

You're now ready to continue with the lab!

**Explore repository**

There is a Git repository named food-scripts consisting of a couple of food-related Python scripts.

Navigate to the repository using the following command:

cd ~/food-scripts

Now, list the files using the ls command. There are three files named **favorite\_foods.log**, **food\_count.py**, and **food\_question.py**.

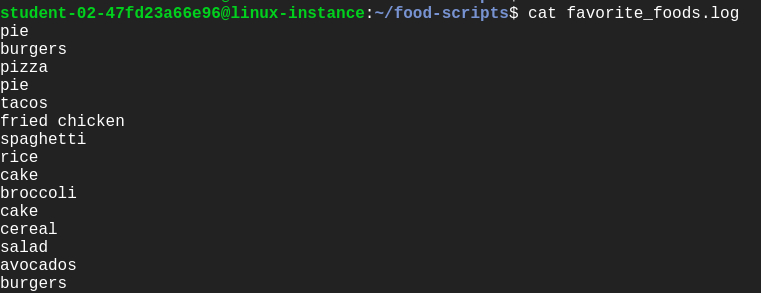
75dd0326f2eee979.png

Let's explore each file. Use the cat command to view each file.

1. **favorite\_foods.log**: This file consists of a list of food items. You can view it using the following command:

cat favorite\_foods.log

Output:

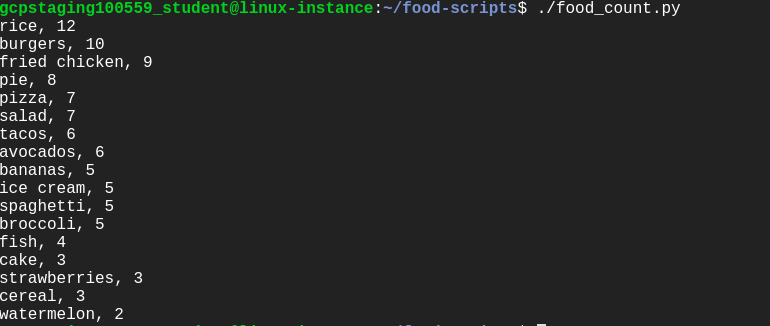


1. **food\_count.py**: This script returns a list of each food and the number of times the food appeared in the favorite\_foods.log file.

Let's execute the script food\_count.py:

./food\_count.py

Output:

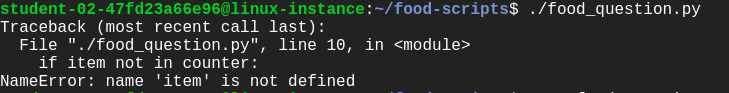


1. **food\_question.py**: This prints a list of foods and prompts the user to enter one of those foods as their favorite. It then returns an answer of how many others in the list like that same food.

Run the following command to see the output of food\_question.py script:

./food\_question.py

Output:



Uh oh , this gives us an error. One of your colleagues reports that this script was working fine until the most recent commit. We'll be fixing this error later during the lab.

**Understanding the repository**

Let's use the following Git operations to understand the workflow of the repository:

* git status
* git log
* git branch

**Git status:** This displays paths that have differences between the index file and the current HEAD commit; paths that have differences between the working tree and the index file; and paths in the working tree that are not tracked by Git. You can view the status of the working tree using the command: [**git status]**

git status

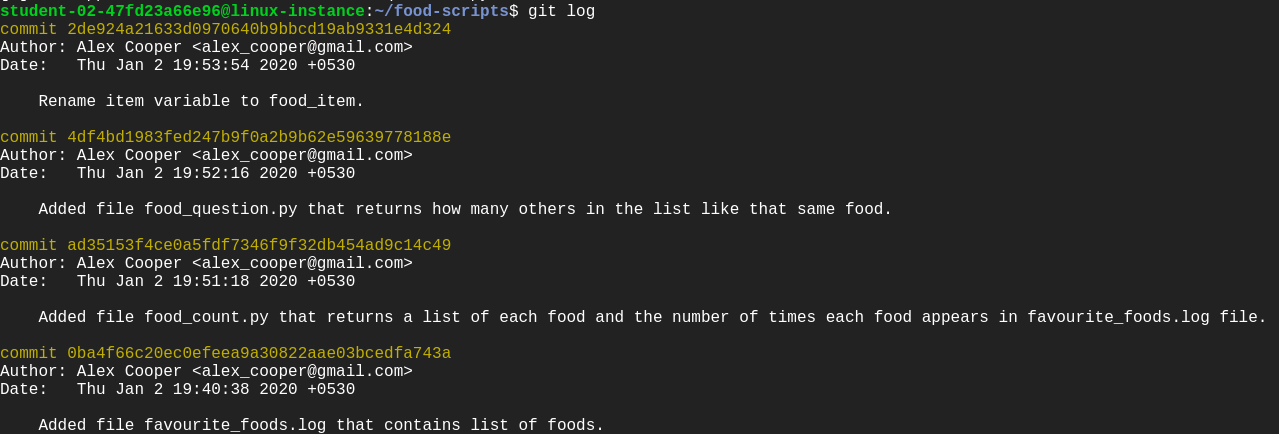
You can now view the status of the working tree.

**Git log:** This lists the commits done in the repository in reverse chronological order; that is, the most recent commits show up first. This command lists each commit with its SHA-1 checksum, the author's name and email, date, and the commit message.

You can see logs by using the following command:

git log

Output:



Enter **q** to exit.

**Git branch:** Branches are a part of the everyday development process on the master branch. Git branches effectively function as a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug, no matter how big or small, you spawn a new branch to encapsulate your changes. This makes it difficult for unstable code to get merged into the main codebase.

Configure Git

Before we move forward with the lab, let's configure Git. Git uses a username to associate commits with an identity. It does this by using the **git config** command. Set the Git username with the following command:

git config user.name "Name"

Replace **Name** with your name. Any future commits you push to GitHub from the command line will now be represented by this name. You can even use **git config** to change the name associated with your Git commits. This will only affect future commits and won't change the name used for past commits.

Let's set your email address to associate them with your Git commits.

git config user.email "user@example.com"

Replace **user@example.com** with your email-id. Any future commits you now push to GitHub will be associated with this email address. You can also use **git config** to change the user email associated with your Git commits.

**Add a new feature**

In this section, we'll be modifying the repository to add a new feature, without affecting the current iteration. This new feature is designed to improve the food count (from the file food\_count.py) output. So, create a branch named **improve-output** using the following command:

git branch improve-output

Move to the improve-output branch from the master branch.

git checkout improve-output

Here, you can modify the script file without disturbing the existing code. Once modified and tested, you can update the master branch with a working code.

Now, open food\_count.py in the nano editor using the following command:

nano food\_count.py

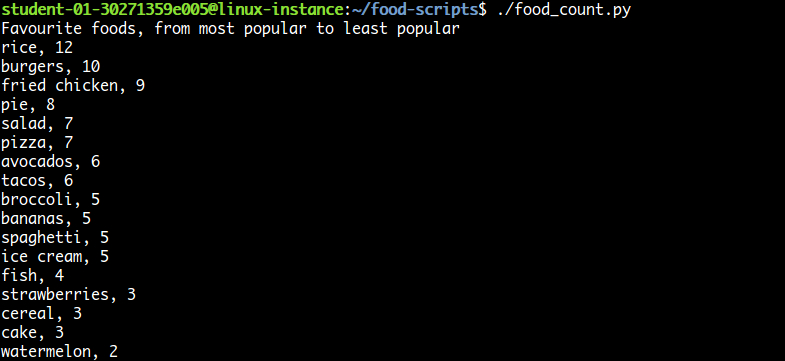
Add the line below before **printing for loop** in the food\_count.py script:

print("Favourite foods, from most popular to least popular")

Save the file by pressing Ctrl-o, the Enter key, and Ctrl-x. Then run the script food\_count.py again to see the output:

./food\_count.py

Output:



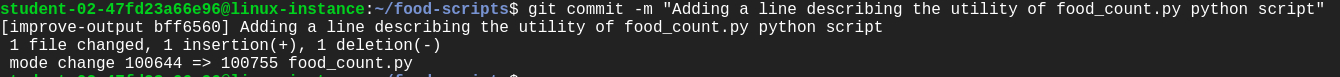
After running the food\_count.py script successfully, commit the changes from the improve-output branch by adding this script to the staging area using the following command:

git add food\_count.py

Now, commit the changes you've done in the improve-output branch.

git commit -m "Adding a line in the output describing the utility of food\_count.py script"

Output:



Click *Check my progress* to verify the objective.

Add a feature

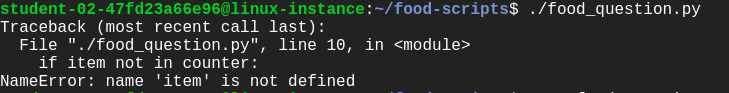
Check my progress

**Fix the script**

In this section, we'll fix the script food\_question.py, which displayed an error when executing it. You can run the file again to view the error.

./food\_question.py

Output:



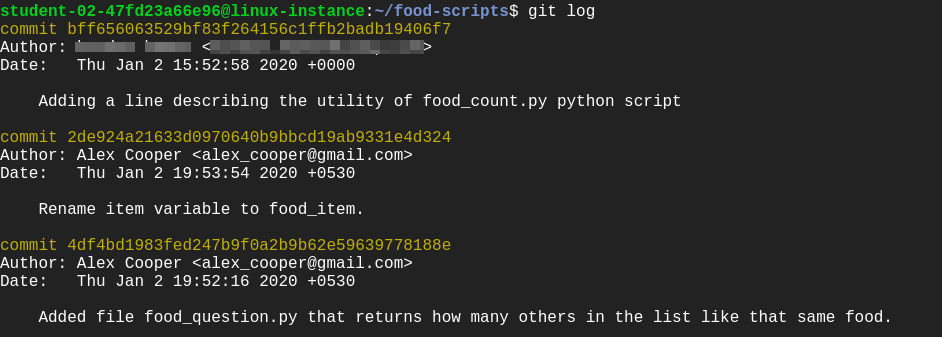
This script gives us the error: "**NameError: name 'item' is not defined**" but your colleague says that the file was running fine before the most recent commit they did.

In this case, we'll revert back the previous commit.

For this, check the git log history so that you can revert back to the commit where it was working fine.

git log

Output:



Here, you'll see the commits in reverse chronological order and find the commit having "**Rename item variable to food\_item**" as a commit message. Make sure to note the commit ID for this particular commit.

Enter **q** to exit.

To revert, use the following command:

git revert [commit-ID]

Replace [commit-ID] with the commit ID you noted earlier.

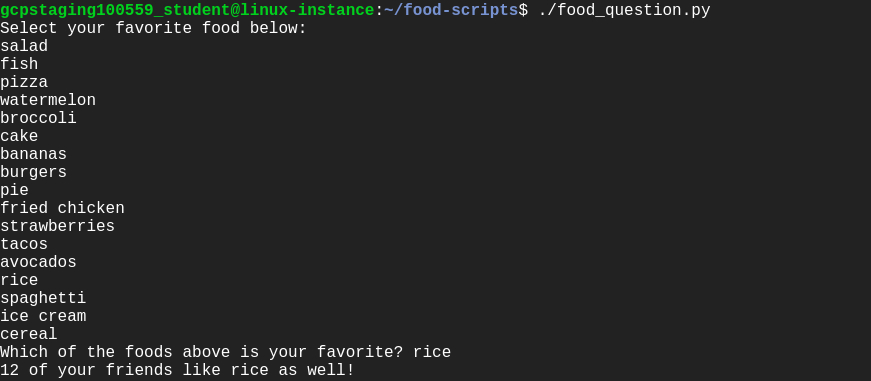
This creates a new commit again. You can continue with the default commit message on the screen or add your own commit message.

Then continue by clicking Ctrl-o, the Enter key, and Ctrl-x.

Now, run food\_question.py again and verify that it's working as intended.

./food\_question.py

Output:



**Merge operation**

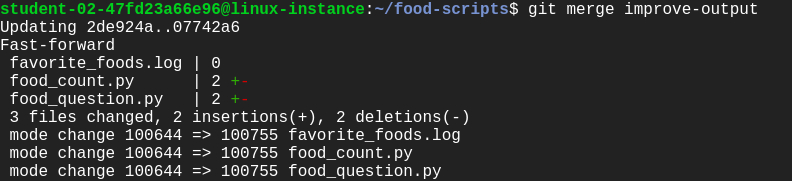
Before merging the branch improve-output, switch to the master branch from the current branch improve-output branch using the command below:

git checkout master

Merge the branch improve-output into the master branch.

git merge improve-output

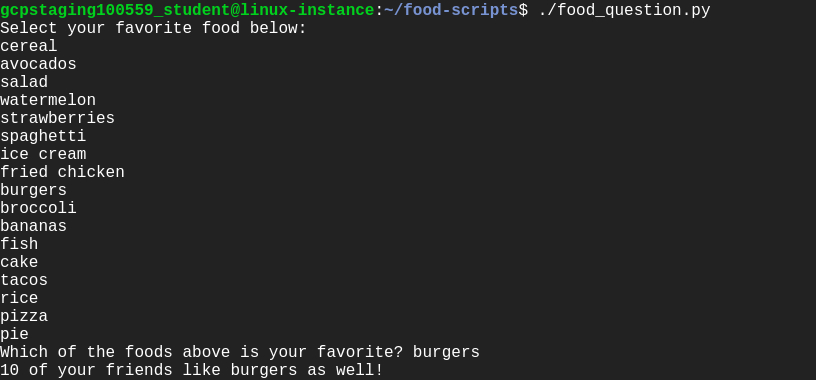
Output:



Now, all your changes made in the improve-output branch are on the master branch.

./food\_question.py

Output:



To get the status from the master branch, use the command below:

git status

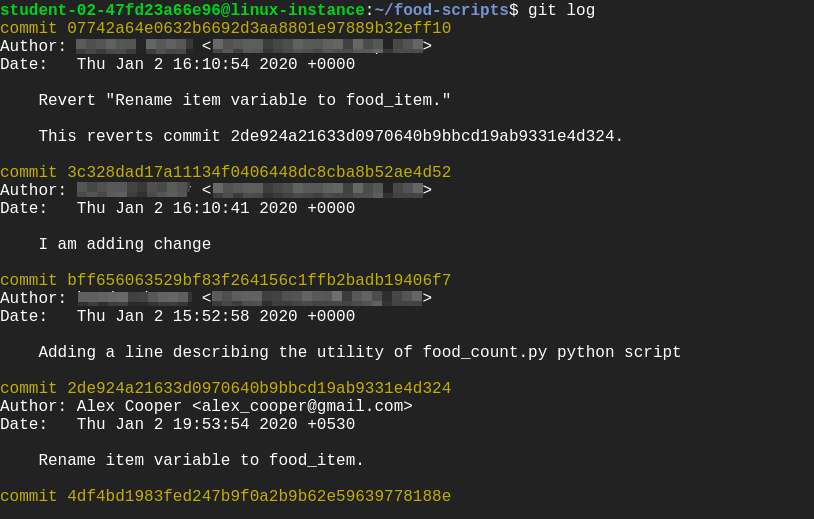
Output:

c18b4c4d4dd08623.png

To track the git commit logs, use the following command:

git log

Output:



Click *Check my progress* to verify the objective.

Revert changes

Check my progress

**Congratulations!**

In this lab, you successfully created a branch from the master branch to add a new feature. You also rolled back a commit to where the script worked fine, and then merged it to the master branch. This will help as you work with colleagues who are simultaneously on the same repository.