



PRIFYSGOL CYMRU
Y Drindod Dewi Sant
UNIVERSITY OF WALES
Trinity Saint David
SWANSEA - ABERTAWE

Applied Computing

Wales Institute of Science and Art (WISA)

Network Programmability (ACCB5029)

Lab 3: Software Version Control with GIT

ICONS Used in this document:



Do research and complete the task. Need referencing/ in-text citations. Shall be included as a part of your coursework.



Record your Reflection



Find out the outcome/ Answers/ Solution

 **Networking**
CISCO™ **Academy**

Task 1:**Objectives**

1. Initializing Git
2. Staging and Committing a File in the Git Repository
3. Managing the File and Tracking Changes
4. Branches and Merging
5. Handling Merge Conflicts
6. Integrating Git with GitHub

Task 2:**In-class Activity****Required Resources**

Host computer with DEVASC VM.

Instructions

Launch the DEVASC VM

1. Initializing Git

- 1.1. Open a terminal in the DEVASC-LABVM. Double-click the Terminal Emulator icon on the desktop.
- 1.2. Initialize a Git Repository.
 - 1.2.1. Use the ls command to display a listing of the current directory. Remember that commands are case-sensitive.
`devasc@labvm:~$ ls`
 - 1.2.2. Configure user information to be used for this local repository.
(This will associate your information the work that you contribute to a local repository. Use your name in place of "Sample User" for the name in quotes ". Use @example.com for your email address.)

Note: These settings can be anything you want at this point. However, when you reset these global values later while integrating Git with GitHub, you will use the username for your GitHub account. If you wish, you can use your GitHub username now.

`devasc@labvm:~$ git config --global user.name "SampleUser"`

`devasc@labvm:~$ git config --global user.email sample@example.com`

- 1.2.3. At any time, you can review these setting with the git config --list command.

```
devasc@labvm:~$ git config --list  
  
user.name=SampleUser
```

user.email=sample@example.com

- 1.2.4. Use the cd command to navigate to the devnet-src folder:

```
devasc@labvm:~$ cd labs/devnet-src/
```

- 1.2.5. Make a directory git-first and change directory into it:

```
devasc@labvm:~/labs/devnet-src$ mkdir git-first
```

What is the command to change into git-first directory?



- 1.2.6. Use the git init command to initialize the current directory (git-first) as a Git repository.

Note: The message displayed indicates that you have created a local repository within your project contained in the hidden directory .git. This is where all your change history is located. You can see it with the ls -a command.

```
devasc@labvm:~/labs/devnet-src/ git-first$ git init
```

Initialized empty Git repository in /home/devasc/labs/devnet-src/ git-first/.git/

```
devasc@labvm:~/labs/devnet-src/ git-first$ ls -a
```

```
. .. .git
```

For your Understanding

As you work on your project, you will want to check to see which files have changed. This is helpful when you are committing files to the repo, and you don't want to commit all of them.

The **git status** command displays modified files in working directory that are staged for your next commit.

This message tells you:

That you are on branch master. (Refer to our discussion on Branches during the lecture)

The commit message is Initial commit.

There is nothing changed to commit.

You will see that the status of your repo will change once you add files and start making changes.

Try this command

```
devasc@labvm:~/labs/devnet-src/ git-first$ git status
```

On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

2. Staging and Committing a File in the Repository

After initializing the Git, we will **create a file, stage that file, and commit that file** to the Git repository.

2.1. Create a file

The git-first repository is created but empty. Using the echo command, create the file NETPRGTY.txt with the information contained in quotes.

```
devasc@labvm:~/labs/devnet-src/git-first$ echo "I am currently enjoying my
group work in Network Programmability Module" > NETPRGTY.txt
```

2.1.1. Use the ls -la command verify the file, as well as the .git directory, are in the git intro directory. Then use cat to display the contents of NETPRGTY.txt.

```
devasc@labvm:~/labs/devnet-src/git-first$ ls -la
```

total 16

```
-rw-rw-r-- 1 devasc devasc 48 Feb 12 05:38 NETPRGTY.txt
```

Note: Refer to previous labs (Review Linux Commands) to understand the commands using here



```
devasc@labvm:~/src/git-first$ cat NETPRGTY.txt
```

I am currently enjoying my group work in Network Programmability module

2.2. Examine the Repository Status.

Examine the repository status using git status. Notice that Git found the new file in the directory and knows that it's not tracked.

```
devasc@labvm:~/labs/devnet-src/git-first$ git status
```

On branch master

No commits yet

Untracked files:

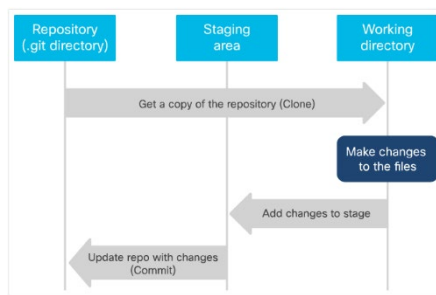
(use "git add <file>..." to include in what will be committed)

```
NETPRGTY.txt
```

nothing added to commit but untracked files present (use "git add" to track)

2.3. Staging the File.

2.3.1. Use the git add command to "stage" the NETPRGTY.txt file.



Staging is an intermediate phase prior to committing a file to the repository with the git commit command. This command creates a snapshot of the contents of the file at the time this command is entered. Any changes to the file require another git add command prior to committing the file.

```
devasc@labvm:~/labs/devnet-src/git-
```

```
first$ git add NETPRGTY.txt
```

2.3.2. Using the git status command again, notice the staged changes displayed as "new file: NETPRGTY.txt".

```
devasc@labvm:~/labs/devnet-src/git-first$ git status
```

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: NETPRGTY.txt

2.4. Committing a File.

Now that you have staged your changes, you will need to commit them to let Git know you want to start tracking those changes.

Commit your staged content as a new commit snapshot by using the git commit command.

The -m message switch enables you to add a message explaining the changes you've made.

Note the number and letter combination highlighted in the output. This is the commit ID. Every commit is identified by a unique SHA1 hash. The commit ID is the first 7 characters of the full commit hash. Your commit ID will be different than the one displayed.

```
devasc@labvm:~/labs/devnet-src/git-first$ git commit -m "Committing  
NETPRGTY.txt to begin tracking changes"
```

```
[master (root-commit) b510f8e] Committing NETPRGTY.txt to begin tracking  
changes
```

1 file changed, 1 insertion(+)

create mode 100644 NETPRGTY.txt

2.5. Viewing the Commit History.

Use the git log command to show all commits in the current branch's history. By default, all commits are made to the master branch. (Branches will be discussed later.)

The first line is the commit hash with the commit ID as first 7 characters. The file is committed to the master branch. This is followed by your name and email address, the date of the commit and the message you included with the commit.

```
devasc@labvm:~/labs/devnet-src/git-first$ git log
```

```
commit b510f8e5f9f63c97432d108a0413567552c07356 (HEAD -> master)
```

```
Author: Sample User <sample@example.com>
```

```
Date: Sun Feb 12 05:43:18 2023 +0000
```

```
Committing NETPRGTY.txt to begin tracking changes
```

3. Modifying the File and Tracking the Changes

In this part, you will modify a file, stage the file, commit the file, and verify changes in the repository.

3.1. Modify the file.

Make a change to NETPRGTY.txt using the echo command. Be sure to use ">>" to append the existing file. The ">" will overwrite the existing file. Use the cat command to view the modified file.

```
devasc@labvm:~/labs/devnet-src/git-first$ echo "I am slowly getting to  
understand Git!" >> NETPRGTY.txt
```

3.1.1. Use the cat command to view the modified file.

```
devasc@labvm:~/labs/devnet-src/git-first$ cat NETPRGTY.txt
```

```
I am currently enjoying my group work in Network Programmability Module
```

```
I am slowly getting to understand Git!
```

3.2. Verify the change to the repository using the git status command.

```
devasc@labvm:~/labs/devnet-src/git-first$ git status
```

```
On branch master
```

```
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: NETPRGTY.txt
```

no changes added to commit (use "git add" and/or "git commit -a")

3.3. Stage the modified file.

The modified file will need to be staged again before it can be committed using the git add command again.

```
devasc@labvm:~/labs/devnet-src/git-first$ git add NETPRGTY.txt
```

3.4. Commit the staged file.

Commit the staged file using the git commit command. Notice the new commit ID.

```
devasc@labvm:~/labs/devnet-src/git-first$ git commit -m "Added additional line to file"
```

```
[master 9f5c4c5] Added additional line to file
```

```
1 file changed, 1 insertion(+)
```

3.5. Verify the changes in the repository.

Use the git log command again to show all commits. Notice that the log contains the original commit entry along with the entry for the commit you just performed. The latest commit is shown first. The output highlights the commit ID (first 7 characters of the SHA1 hash), the date/time of the commit, and the message of the commit for each entry.

```
devasc@labvm:~/labs/devnet-src/git-first$ git log
```

```
commit 9f5c4c5d630e88abe2a873fe48144e25ebe7bd6a (HEAD -> master)
```

```
Author: Sample User <sample@example.com>
```

```
Date: Sun Feb 12 05:54:18 2023 +0000
```

```
Added additional line to file
```

```
commit b510f8e5f9f63c97432d108a0413567552c07356
```

```
Author: Sample User <sample@example.com>
```

```
Date: Sun Feb 12 05:59:26 2023 +0000
```

```
Committing NETPRGTY.txt to begin tracking changes
```

3.5.1. When you have multiple entries in the log, you can compare the two commits using the git diff command adding original commit ID first and the latest commit second: git diff <commit ID original> <commit ID latest>. You will need to use your commit IDs. The "+" sign at the end, followed by the text indicates the content that was appended to the file.

```
devasc@labvm:~/labs/devnet-src/git-first$ git diff b510f8e 9f5c4c5
```

```
diff --git a/NETPRGTY.txt b/NETPRGTY.txt
```

```
index 93cd3fb..085273f 100644
```

```
--- a/NETPRGTY.txt
```

```
+++ b/NETPRGTY.txt
```

```
@@ -1 +1,2 @@
```

```
I am currently enjoying my group work in Network Programmability Module
```

```
+I am slowly getting to understand Git!
```

4. Branches and Merging

When a repository is created, the files are automatically put in a branch called master. Whenever possible it is recommended to use branches rather than directly updating the master branch. Branching is used so that you can make changes in another area without affecting the master branch. This is done to help prevent accidental updates that might overwrite existing code.

Now, we will create a new branch, checkout the branch, make changes in the branch, stage and commit the branch, merge the branch changes to the master branch, and then delete the branch.

4.1. Create a new branch

Create a new branch called feature using the git branch <branch-name> command.

```
devasc@labvm:~/labs/devnet-src/git-first$ git branch feature
```

4.2. Verify current branch

Use the git branch command without a branch-name to display all the branches for this repository. The "*" next to the master branch indicates that this is the current branch – the branch that is currently "checked out".

```
devasc@labvm:~/labs/devnet-src/git-first$ git branch
```

```
feature
```

```
* master
```

4.3. Checkout the new branch

Use the git checkout <branch-name> command to switch to the feature branch.

```
devasc@labvm:~/labs/devnet-src/git-first$ git checkout feature
```

4.4. Verify current branch

Verify you have switched to the feature branch using the git branch command. Note the "*" next to the feature branch.

What is the command to verify that you have switched to the feature branch?



Are you getting the below output after the command issued?

* feature

Master

- 4.5. Append a new line of text "This text was added originally while in the feature branch" to the NETPRGTY.txt file, again using the echo command and verify the appended line.



Command to Append

Command to verify

Output expected:

I am currently enjoying my group work in Network Programmability Module

I am beginning to understand Git!

This text was added originally while in the feature branch

- 4.6. Stage the modified file in the feature branch. Stage the updated file to the current feature branch and confirm the file is staged.



Command to Stage:

Command to confirm the file is staged:

Output expected:

On branch feature

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: NETPRGTY.txt

- 4.7. Commit the staged file in the feature branch. Commit the staged file using the git commit command with a note "Added a third line in feature branch". Notice the new commit ID and your message.



Command to commit the staged file:

Output expected:

[feature cd828a7] Added a third line in feature branch

1 file changed, 1 insertion(+)

- 4.8. Confirm and show all commits including the commit you just did to the feature branch. The prior commit was done within the master branch.



Command to commit the staged file:

Output expected:

```
commit cd828a73102cf308981d6290113c358cbd387620 (HEAD -> feature)
```

```
Author: Sample User <sample@example.com>
```

```
Date: Sat Apr 18 22:59:48 2020 +0000
```

```
Added a third line in feature branch
```

```
commit 9f5c4c5d630e88abe2a873fe48144e25ebe7bd6a (master)
```

```
Author: Sample User <sample@example.com>
```

```
Date: Sat Apr 18 19:17:50 2020 +0000
```

```
Added additional line to file
```

```
commit b510f8e5f9f63c97432d108a0413567552c07356
```

```
Author: Sample User sample@example.com
```

```
Date: Sat Apr 18 18:03:28 2020 +0000
```

```
Committing NETPRGTY.txt to begin tracking changes
```

- 4.9. Switch to the master branch using the appropriate command and verify the current working branch using the git branch command.



Command to switch:

Command to verify:

Expected Output:

```
feature
```

```
* master
```

- 4.10. Merge file contents from feature to master branch.

Branches are often used when implementing new features or fixes. They can be submitted for review by team members, and then once verified, can be pulled into the main codebase – the master branch.

Merge the contents (known as the history) from the feature branch into the master branch using the git merge <branch-name> command. The branch-name is the branch that histories are pulled from into the current branch. The output displays that one file was changed with one line inserted.

```
devasc@labvm:~/labs/devnet-src/git-first$ git merge feature
```

Updating 9f5c4c5..cd828a7

Fast-forward

NETPRGTY.txt | 1 +

1 file changed, 1 insertion(+)

- 4.11. Verify the appended content to the NETPRGTY.txt file in the master branch using the cat command.

```
devasc@labvm:~/labs/devnet-src/git-first$ cat NETPRGTY.txt
```

I am currently enjoying my group work in Network Programmability Module

I am slowly getting to understand Git!

This text was added originally while in the feature branch

Hope, you are now clear to create a file, work on a file, create branches, stage the changes, and commit them to the repository. Let us delete the branch and do a new task.

- 4.12. Verify the feature branch is still available.

Command to verify:

Expected Output:

feature

* master



- 4.13. Delete the feature branch using the git branch -d <branch-name> command. .

```
devasc@labvm:~/labs/devnet-src/git-first$ git branch -d feature
```

Deleted branch feature (was cd828a7).

- 4.14. Again, verify the feature branch is no longer available command.

Command to verify:

Expected Output:

* master



5. Handling Merge Conflicts

At times, you may experience a merge conflict. This is when you may have made overlapping changes to a file, and Git cannot automatically merge the changes.

Now, let us create a test branch, modify its content, stage and commit the test branch, switch to the master branch, modify the content again, stage and commit the master branch, attempt to merge branches, locate and resolve the conflict, stage and commit the master branch again, and verify your commit.

- 5.1. Create a new branch test.

Command:



- 5.2. Switch to the branch test.

Command:

- 5.3. Verify the working branch is the test branch.



Command:

Expected Outcome:

master

* test

- 5.4. Verify the current contents of NETPRGTY.txt. Notice the first line includes the word "Module".

Command:



Expected Outcome:

I am currently enjoying my group work in Network Programmability Module

I am slowly getting to understand Git!

This text was added originally while in the feature branch

- 5.5. Modify the contents of NETPRGTY.txt in the test branch.

Use the sed command to change the word "Module" to "Concepts" in the NETPRGTY.txt file.

```
devasc@labvm:~/labs/devnet-src/git-first$ sed -i 's/Cisco/Concepts/'
NETPRGTY.txt
```

Discuss about similar commands that can do the same action?



- 5.6. Verify the contents of the modified NETPRGTY.txt in the test branch.

Command:

Expected Outcome:

I am currently enjoying my group work in Network Programmability Concepts

I am slowly getting to understand Git!

This text was added originally while in the feature branch

5.7. Stage and commit the test branch.

Stage and commit the file with a single git commit -a command. The -a option only affects files that have been modified and deleted. It does not affect new files.

```
devasc@labvm:~/labs/devnet-src/git-first$ git commit -a -m "Change Module to Concepts"
```

```
[test b6130a6] Change Module to Concepts
```

```
1 file changed, 1 insertion(+), 1 deletion(-)
```

5.8. Checkout the master branch. Switch to the master branch.

Command:

Expected Outcome:

Switched to branch 'master'



5.9. Verify that the master branch is your current working branch.

Command:

Expected Outcome:

```
* master
```

```
test
```



5.10. Modify the contents of NETPRGTY.txt in the master branch using the sed command. Change the word "Module" to "ACCB5029" in the NETPRGTY.txt file.

Command:



5.11. Verify the contents of the modified NETPRGTY.txt in the master branch.

Command:



Expected Outcome:

I am currently enjoying my group work in Network Programmability
ACCB5029

I am slowly getting to understand Git!

This text was added originally while in the feature branch

- 5.12. Stage and commit the master branch using the git commit -a command with comments "Changed Module to ACCB5029".

Command:



Expected Outcome:

```
[master 72996c0] Changed Module to ACCB5029
```

```
1 file changed, 1 insertion(+), 1 deletion(-)
```

- 5.13. Attempt to merge the test branch history into the master branch.

```
devasc@labvm:~/labs/devnet-src/git-first$ git merge test
```

Auto-merging NETPRGTY.txt

CONFLICT (content): Merge conflict in NETPRGTY.txt

Automatic merge failed; fix conflicts and then commit the result.

```
devasc@labvm:~/labs/devnet-src/git-first$
```

5.14. Working with conflicts.

- 5.14.1. Use the git log command to view the commits. Notice that the HEAD version is the master branch. This will be helpful in the next step.

```
devasc@labvm:~/labs/devnet-src/git-first$ git log
```

```
commit 72996c09fa0ac5dd0b8ab9ec9f8530ae2c5c4eb6 (HEAD -> master)
```

```
Author: Sample User <sample@example.com>
```

```
Date: Sun Feb 12 06:25:07 2023 +0000
```

```
Changed Module to ACCB6019
```

```
<output omitted>
```

- 5.14.2. Use the cat command to view the contents of the NETPRGTY.txt file. The file now contains information to help you find the conflict. The HEAD version (master branch) containing the word "DevNet" is conflicting with the test branch version and the word "NetAcad".

```
devasc@labvm:~/labs/devnet-src/git-first$ cat NETPRGTY.txt
```

```
<<<<<<< HEAD
```

```
I am currently enjoying my group work in Network Programmability
accb6019
```

```
=====
```

I am currently enjoying my group work in Network Programmability concepts

```
>>>>>> test
```

I am beginning to understand Git!

This text was added originally while in the feature branch

5.14.3. Manually edit the NETPRGTY.txt file to remove the conflicting text.

Use the vim command to edit the file.

```
devasc@labvm:~/labs/devnet-src/git-first$ vim NETPRGTY.txt
```

Use the up and down arrow to select the proper line of text. Press dd (delete) on the following lines that are highlighted. dd will delete the line the cursor is on.

```
<<<<<<< HEAD
```

I am currently enjoying my group work in Network Programmability accb6019

```
=====
```

I am currently enjoying my group work in Network Programmability Module

```
>>>>>> test
```

I am beginning to understand Git!

This text was added originally while in the feature branch

Save your changes in vim by pressing ESC (the escape key) and then typing : (colon) followed by wq and press enter.

5.15. Verify your edits of NETPRGTY.txt in the master branch.

Command:



Expected Outcome:

I am currently enjoying my group work in Network Programmability accb6019

I am beginning to understand Git!

This text was added originally while in the feature branch

5.16. Stage and commit the master branch.

Stage and commit NETPRGTY.txt to the master branch using the git commit -a command with a comment "Manually merged from test branch". Note: You must add before committing.

Command:



Expected Outcome:

[master 22d3da4] Manually merged from test branch

- 5.17. Verify the commit. Use the git log command to verify the commit. If necessary, you can use q to quit out of the git log display.

Command:



Expected Outcome

commit 22d3da41e00549ce69dc145a84884af6a1697734 (HEAD -> master)

Merge: 729496c0 b6130a6

Author: Sample User <sample@example.com>

manually merged from branch test

<output omitted>

6. Integrating Git with GitHub

So far, all the changes you have made to your file have been stored on your local machine. Git runs locally and does not require any central file server or cloud-based hosting service. Git allows a user to locally store and manage files.

Although Git is useful for a single user, integrating the local Git repository with a cloud-based server like GitHub is helpful when working within a team. Each team member keeps a copy on the repository on their local machine and updates the central cloud-based repository to share any changes.

There are quite a few popular Git services, including GitHub, Stash from Atlassian, and GitLab. Because it is readily accessible, you will use GitHub in these examples.

- 6.1. Create a GitHub Account/ Log into your GitHub Account Create a Repository.
- 6.2. Create a Repository.
Select the "New repository" button or click on the "+" icon in the upper right corner and select "New repository".
- 6.2.1. Create a repository using the following information:

Repository name: np-study-team

Description: Trail to work together for Group Work Task

Public/Private: Private

Select: Create repository

- 6.3. Create a new directory np-study-team.
If you are not already in the git-first directory, change to it now.

```
devasc@labvm:~$ cd ~/labs/devnet-src/git-first
```

- 6.3.1. Make a new directory called np-study-team. The directory does not have to match the name as the repository.

```
devasc@labvm:~/labs/devnet-src/git-first$ mkdir np-study-team
```

- 6.3.2. Change directory to np-study-team. Use the cd command to change directories to np-study-team.

Command:



Expected Outcome

```
devasc@labvm:~/labs/devnet-src/git-first/np-study-team $
```

- 6.4. Copy and verify the file.

- 6.4.1. Use the cp command to copy the NETPRGTY.txt from git-first parent directory to the np-study-team sub-directory.

Command:



Expected Outcome

```
devasc@labvm:~/labs/devnet-src/myGit-first/np-study-team$ ls  
NETPRGTY.txt
```

- 6.4.2. Verify the file contents.

Command:



Expected Outcome

I am currently enjoying my group work in Network Programmability concepts

I am slowly getting to understand GIT

- 6.5. Initialize a new Git repository.

Use the git init command to initialize the current directory (np-study-team) as a Git repository. The message displayed indicates that you have created a local repository within your project contained in the hidden directory .git. This is where all your change history is located.

```
devasc@labvm:~/labs/devnet-src/git-first/np-study-team $ git init
```

Initialized empty Git repository in `/home/devasc/src/git-first/np-study-team/.git/`

- 6.6. Next, check your global git variables.

Command:



Expected Outcome

```
user.name=SampleUser
```

```
user.email=sample@example.com
```

```
core.repositoryformatversion=0
```

```
core.filemode=true
```

```
core.bare=false
```

```
core.logallrefupdates=true
```

- 6.7. If the `user.name` and `user.email` variables do not match your GitHub credentials, change them now.

```
devasc@labvm:~$ git config --global user.name "GitHub username"
```

```
devasc@labvm:~$ git config --global user.email GitHub-email-address
```

- 6.8. Point Git repository to GitHub repository.

Use the `git remote add` command to add a Git URL as a remote alias. The value "origin" points to the newly created repository on GitHub. Use your GitHub username in the URL path for `github-username`. Note: Your username is case-sensitive.

```
devasc@labvm:~/labs/devnet-src/git-first/np-study-team $ git remote add origin https://github.com/github-username/np-study-team .git
```

- 6.9. Verify the remote is running on github.com.

```
devasc@labvm:~/labs/devnet-src/git-first/np-study-team $ git remote --verbose
```

```
origin https://github.com/username/np-study-team .git (fetch)
```

```
origin https://github.com/username/np-study-team .git (push)
```

- 6.10. View the git log. The error indicates that there are no commits.

```
devasc@labvm:~/labs/devnet-src/git-first/np-study-team $ git log
```

fatal: your current branch 'master' does not have any commits yet

- 6.11. Stage and commit the NETPRGTY.txt file. Use the git add command to stage the NETPRGTY.txt file.

```
devasc@labvm:~/labs/devnet-src/git-first/np-study-team $ git add NETPRGTY.txt
```

- 6.12. Use git commit command to commit the NETPRGTY.txt file.

```
devasc@labvm:~/labs/devnet-src/git-first/np-study-team $ git commit -m "Add NETPRGTY.txt file to np-study-team "
```

```
[master (root-commit) c60635f] Add NETPRGTY.txt file to np-study-team
```

```
1 file changed, 3 insertions(+)
```

```
create mode 100644 NETPRGTY.txt
```

- 6.13. Verify the commit.

Command:



Expected Outcome

```
commit c60635fe4a1f85667641afb9373e7f49a287bdd6 (HEAD -> master)
```

```
Author: username <user@example.com>
```

```
Add NETPRGTY.txt file to np-study-team
```

- 6.14. Use the git status command to view status information. The phrase "working tree clean" means that Git has compared your file listing to what you've told Git, and it's a clean slate with nothing new to report.

```
devasc@labvm:~/labs/devnet-src/git-first/np-study-team $ git status
```

```
On branch master
```

```
nothing to commit, working tree clean
```

- 6.15. Upload the file by logging in to your GitHub page.
6.15.1. Click on Code, Add files via Upload. Check if the files are uploaded.

TASK – 2

In-class Activity: Total Time **60 Minutes**

Challenge-driven:

Today, you will be doing the Lab 3 – Task 1 individually. Continue with Task 2 by working in Groups. You are free to form your own team.

Why and What is this Challenge-driven activity?

As an individual,

- ✓ you will be able to work with your team on the given challenge (Lab 3 – Task 2) with an open mind, while adopting the ability to critique and challenge each other's thinking within the area of discussion. You will be able to analyse and evaluate information that you present to others and vice versa.
- ✓ With the help of constructive criticism from your teammates and others, you will be able to point out the areas that require more focus to develop your problem-solving skills with confidence.
- ✓ By interacting with your fellow classmates, the ideas from each others can evidently lead to innovative solutions leading you to think out of the box with the ability to take calculated risk.

Assessment Criteria's:

1. Group Dynamics: Mixed group
2. Time Management: Complete within Time
3. Behavioural Management: well-behaving approach, team etiquettes, etc
4. Outcome for the challenge

Challenge:

Work as a Team and complete sending another new file collaborated by all the team members with different version changes (make sure all the team members are involved) from Git (push) to GitHub without using Upload option (task completed in Lab 3).

Use the git push origin master command to send (push) the file to your GitHub repository. When authenticating the git operations to GitHub, you need to use a Personal Access Token instead of using your GitHub account password. You can generate a new Token at your GitHub profile settings page with a full "repo" scope options and use it when asked for your GitHub password.

Document the steps with necessary screenshots and submit the file in Moodle with reflections recorded accordingly. Don't forget to verify file on GitHub.

Team Members & Roles

Team Roles	Name	Contribution

Before the activity: (Pre-work)

Before working on this challenge as a team, recall our lectures and do research on Software Development Methodologies. Based on your research,

- Which methodology do you think is the best in today's Software Development Scenario?
- Based on the task completed last week (Lab 2), What is the best Software Development approach you think will be efficient to work as a Team? Why?

During the activity:

Planning: (10-15 Minutes)

Discuss and include all your plan to complete this challenge. Address questions like (not limited to)

- what files you are going to create?
- How to work on those files?
- Do all the team members have a GitHub account?
- How to collaborate?
- What is the timeline for each task?
- Who is responsible for what?
- How are we (as a team) going to fit into the above discussed software development approach?
- Etc.

Note: You are given 60 minutes time to complete this challenge. 15 minutes gone already in Pre-work. During the 15 minutes of planning, consider how you are going to set up this task and complete within 45 minutes? Allocate time for each part including documentation and reflection.

Implementation & Documentation:(Minutes)

Complete the challenge given as a Team. Include necessary screenshots (all your team) and give adequate description. Note: Readers of this document would be your lectures, other team members, employers, even students doing this module in future. *(However, your names and group details will not be shared)* Make sure, the documentation is adequate to provide clear understanding on the task completed.

Reflection (Minutes)

Reflection on the task and the activity from each member followed by a summary of this work as a group

Reflect on

- ✓ How your team designed this task?
- ✓ What is your contribution (Your names (if given) will be anonymised as Contributor 1, Contributor 2 etc)
- ✓ What did you learn from this task?
- ✓ What was really challenging in the task?
- ✓ Did you come across any challenges in completing the task? If so, how did you overcome the challenge?
- ✓ Did the task work as plan?
- ✓ Did all the team members collaborated and cooperated? If not, what actions was taken
- ✓ (Not limited to)

After the activity: (Post work)

+ Research

We discussed about 3 types of version control systems. During the tutorials and in-class activity, we were working on Git (as a Local/ Distributed version control tool) and GitHub as cloud repo. Do research on what other tools like GIT is available for Local, Centralized and Distributed VCS.

+ Summary

Write your reflections about the challenge driven task. Consider reflecting how group dynamics, time management, behavioural management created an impact on the outcome of the challenge.

End of document.