Git for beginners - Handbook

(An open source distributed version control system)

Prepared by – Anitha Ramesh

Talent Transformation, Wipro Technologies

About Version control Tools

What is Version Control?

From: https://en.wikipedia.org/wiki/Version_control

A component of software configuration management, version control, also known as revision control or source control, is the management of changes to documents, computer programs, large web sites, and other collections of information.

Changes are usually identified by a number or letter code, termed the "revision number", "revision level", or simply "revision". For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on. Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

Reasons for Version Control Systems

Use case 1: Maintaining history

- The life of your software is recorded from the beginning
- At any moment, you can revert to a previous revision in case if you are not happy with the latest changes
- You can browse through the history and understand
  - ➢ When was it done?
  - ➢ Who has made the change?
  - ➢ What was the change?
  - ➢ In which context?

Use case 2: Working with others

- VC Tools helps you to share files within your team
- Merge changes done by others
- Ensure that nothing is actually overwritten

Use case 3: Branching

You can have multiple variants of the same software, materialized as branches

- a main branch
- a maintenance branch (to provide bug fixes for earlier release)
- a development branch
- a release branch (to freeze code before a new release)

History

- before 2005: Linux sources were managed with Bitkeeper
- April 2005: revocation of free-use license happened
- Version control tools available in the market were not mature enough to meet the performance constraints

- **Linus Torvalds** started developing Git
- December 2005:Git 1.0 released
- Today, Git is one of the most popular distributed version control tool available in the market

Distributed version control is a form of version control where the complete codebase - including its full history is mirrored on every developer's computer

Version control tools that are popular today

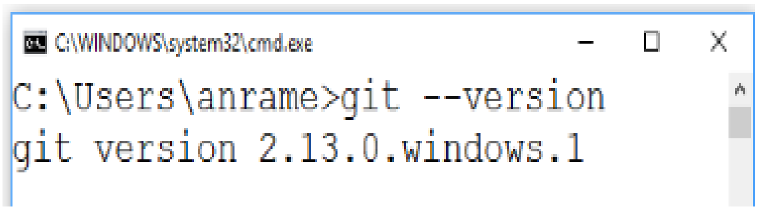
Bitbucket and GitHub are two of the most popular version control system management services.

Github is a mature, actively maintained open source project and it enables an unlimited number of collaborators for public and private repositories.

Bitbucket offers free private repositories for up to five collaborators.

Git installation

You can open the command prompt and check if git is already installed in your system, using git –version command



```
C:\Users\anrame>git --version
git version 2.13.0.windows.1
```

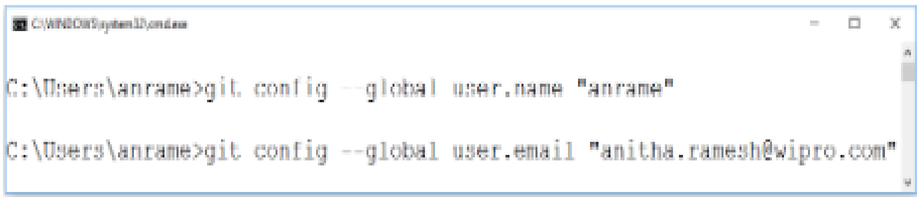
- If it is not installed, for windows users, download git from the below url and install the same, following the instructions. The default options are sufficient for most users.

  https://gitforwindows.org/

Your identity

The first thing you should do when you install Git is to set your user name and email address. This is important because every Git commit uses this information.

You need to provide your user name and your email id, instead of the one's shown below



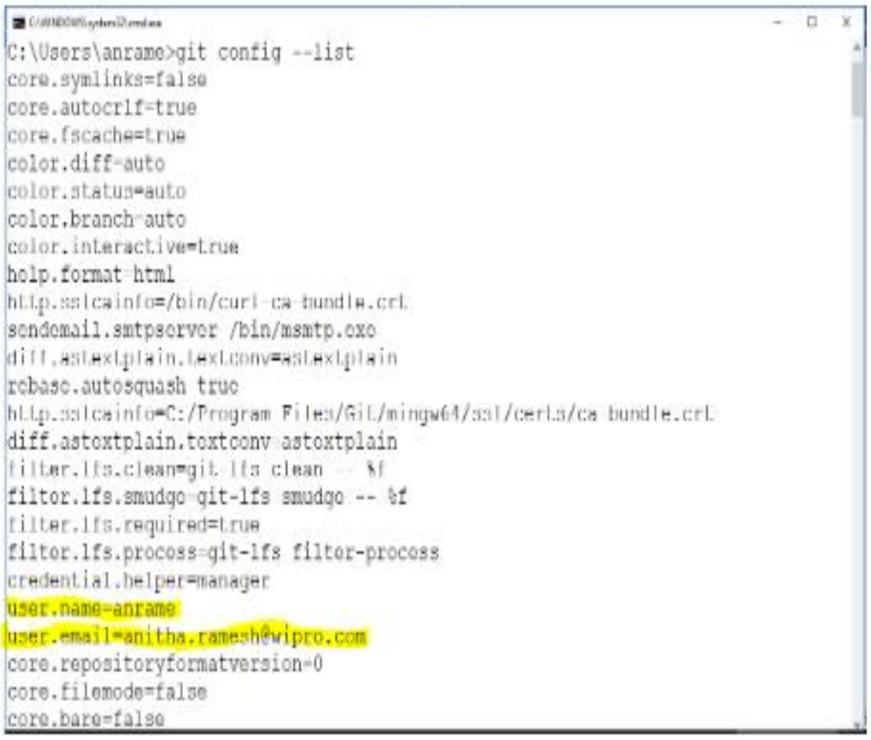
```
C:\Users\anrame>git config --global user.name "anrame"

C:\Users\anrame>git config --global user.email "anitha.ramesh@wipro.com"
```

Again, you need to do this only once if you pass the --global option, because then Git will always use that information for anything you do on that system. If you want to override this with a different name or email address for specific projects, you can run the command without the --global option when you are in that project.

Checking your configuration settings

If you want to check your configuration settings, you can use the

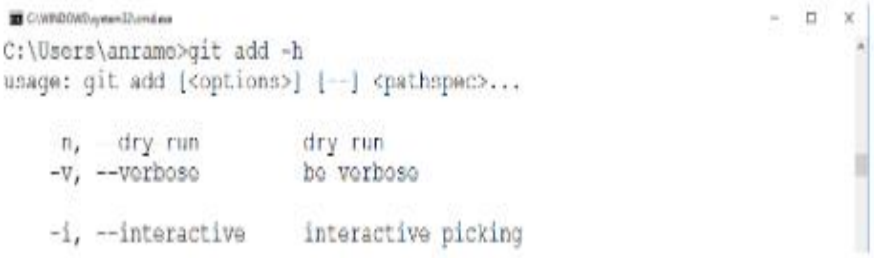
git config --list command



```
C:\Users\anrame>git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
http.sslcainfo=/bin/curl-ca-bundle.crt
sendemail.smtpserver=/bin/msmtp.exe
diff.astextplain.textconv=astextplain
rebase.autosquash=true
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.required=true
filter.lfs.process=git-lfs filter-process
credential.helper=manager
user.name=anrame
user.email=anitha.ramesh@wipro.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
```

Getting help

You can use –h option for getting help on a particular command



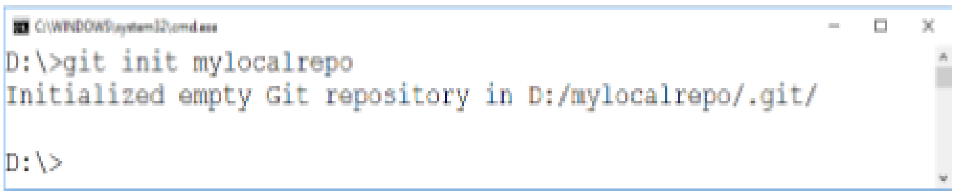
```
C:\Users\anramo>git add -h
usage: git add [<options>] [--] <pathspec>...

    -n, --dry-run         dry run
    -v, --verbose         be verbose

    -i, --interactive     interactive picking
```

Working Locally

Creating a new repository

A git repository is a virtual storage of your project. It allows you to save versions of your code, which you can access later.

git init mylocalrepo



```
D:\>git init mylocalrepo
Initialized empty Git repository in D:/mylocalrepo/.git/

D:\>
```

The init command stands for initialize. It creates a new Git reposotory. Once you run "git init", Git will initialize a hidden directory called ".git" in the project's root directory.

This command creates the directory mylocalrepo

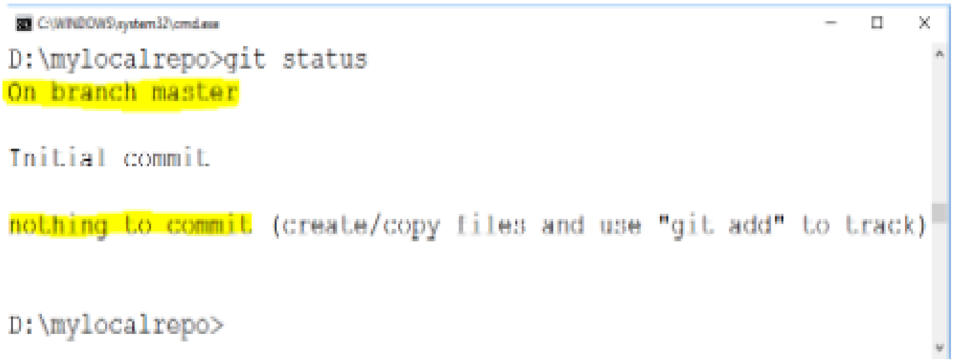
The repository is located in mylocalrepo/.git

The /.git/ directory contains your whole history, so do not delete it

The (initially empty) working copy is located in mylocalrepo

Know your git status

You can move to the mylocalrepo and give ***git status*** command to know the status of your repository

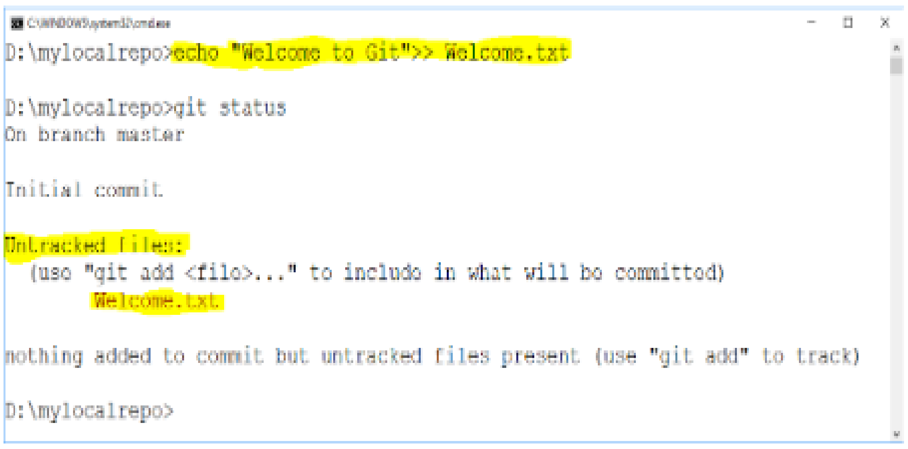


```
D:\mylocalrepo>git status
On branch master

Initial commit

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


D:\mylocalrepo>
```

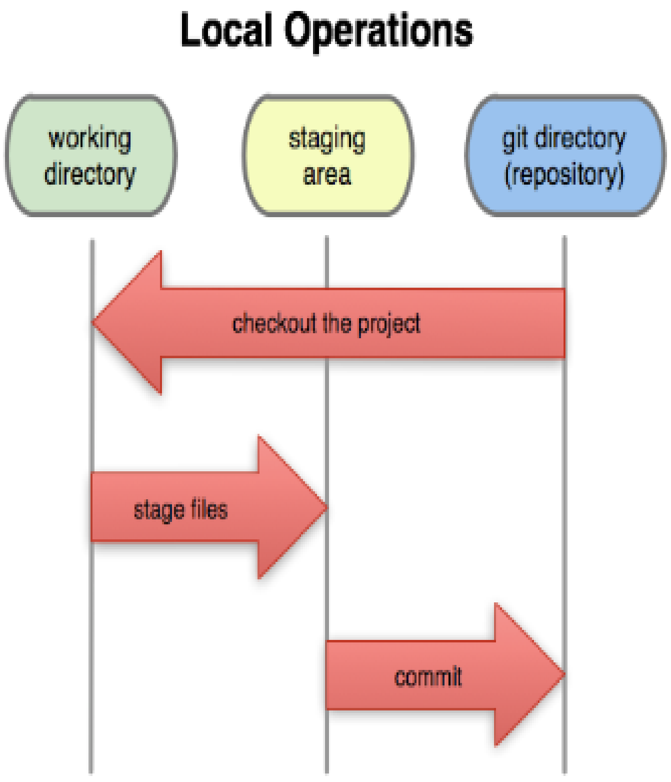
master is the default branch created by git. Since there are no files in the directory yet, it says 'nothing to commit'.

Add files to the repository

You can add new files to the repository and check the status



```
C:\WINDOWS\system32\cmd.exe
D:\mylocalrepo>echo "Welcome to Git">> Welcome.txt

D:\mylocalrepo>git status
On branch master

Initial commit

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

        Welcome.txt

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

D:\mylocalrepo>
```



Your local repository consists of three trees maintained by git. The first one is your working directory, which holds the actual files. The second one is the index, which acts as a staging area and finally the head which points to the last commit you have made.

Whenever there are any changes done in the work directory, the changes need to be first added to the staging area using a *git add* command and from staging area, the files need to be moved to the git repository using *git commit.*

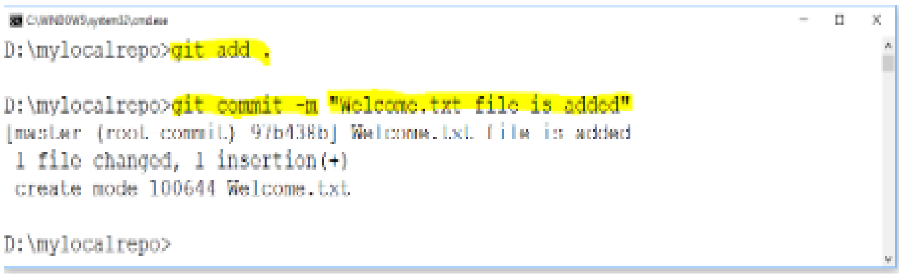
Add and Commit your files

- **Single/Multiple file**

  git add file1 [file2..]

  git commit file1 [file2..]

- **All files in the current directory**

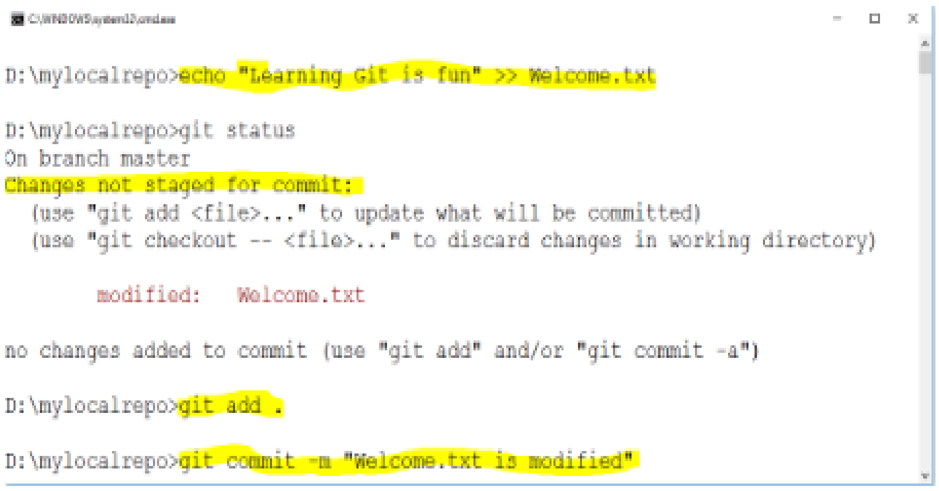


```
C:\WINDOWS\system32\cmd.exe
D:\mylocalrepo>git add .

D:\mylocalrepo>git commit -m "Welcome.txt file is added"
[master (root-commit) 97b438b] Welcome.txt file is added
 1 file changed, 1 insertion(+)
 create mode 100644 Welcome.txt

D:\mylocalrepo>
```

The *git add* . command will add all files from working directory and *git commit* will commit all the files to the local git repository.

Whenever you commit the files, you need to add –m option and provide some comments about the commit.

Update a file



```
D:\mylocalrepo>echo "Learning Git is fun" >> Welcome.txt

D:\mylocalrepo>git status
On branch master
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:   Welcome.txt

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

D:\mylocalrepo>git add .

D:\mylocalrepo>git commit -m "Welcome.txt is modified"
```

Whenever some files in the working direcotry are modified, you need to again add them to the staging area and commit those.

Bypassing the staging area

Running git add and git commit for every iteration is tedious.

GIT provides a way to bypass the staging area

git commit file1 [file2 ..]

This command commits files (or dirs) directly from the working tree

Note: when bypassing the index, git ignroes new files

git commit . - commits only files that were present in the last commit (updated files)

git add . && git commit - commits everything in the working tree (including new files)

Interacting with a remote repository

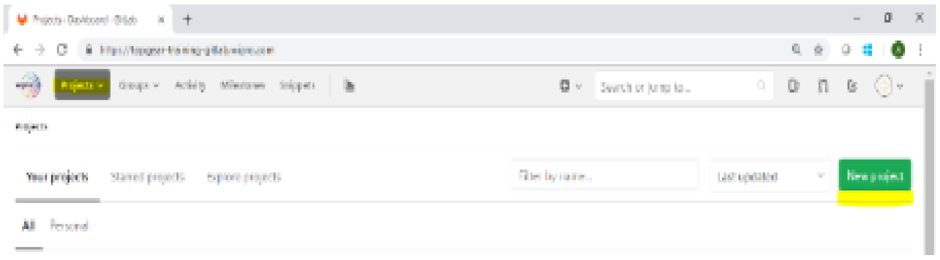
Within Wipro, for remote repository, we need to use gitlab instead of github.

The following is the url for topgear training gitlab

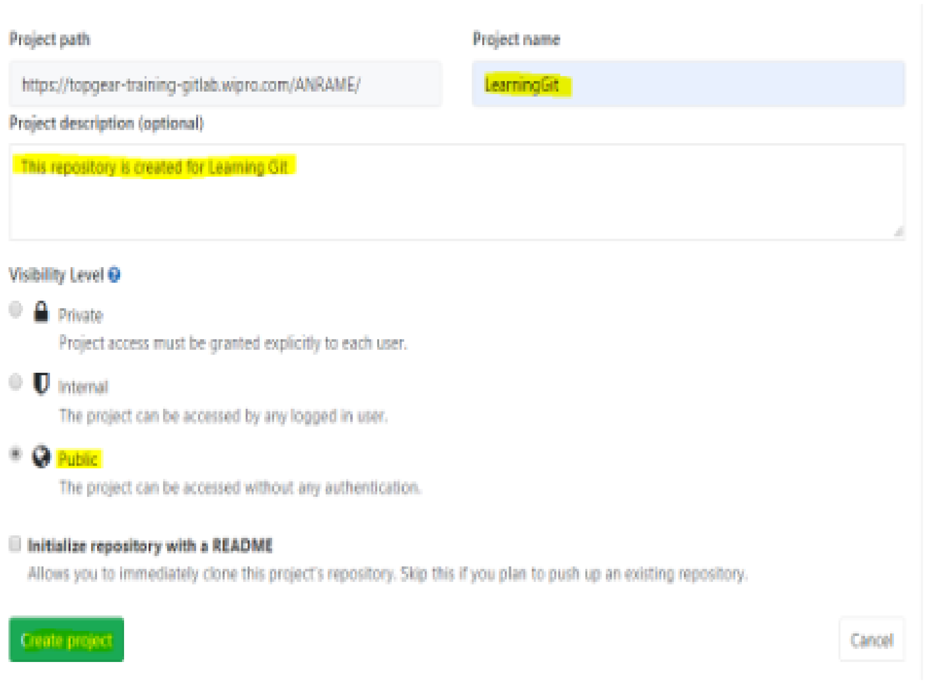
https://topgear-training-gitlab.wipro.com/users/sign_in

Sign in to this url, with your Wipro ADID and password

Create a new repository in gitlab, by clicking on New Project



After clicking the New Project button, gitlab will ask you to name your repo and provide a brief description. You can choose the visibility level of your repository later



Project path
https://topgear-training-gitlab.wipro.com/ANRAME/

Project name
LearningGit

Project description (optional)
This repository is created for Learning Git

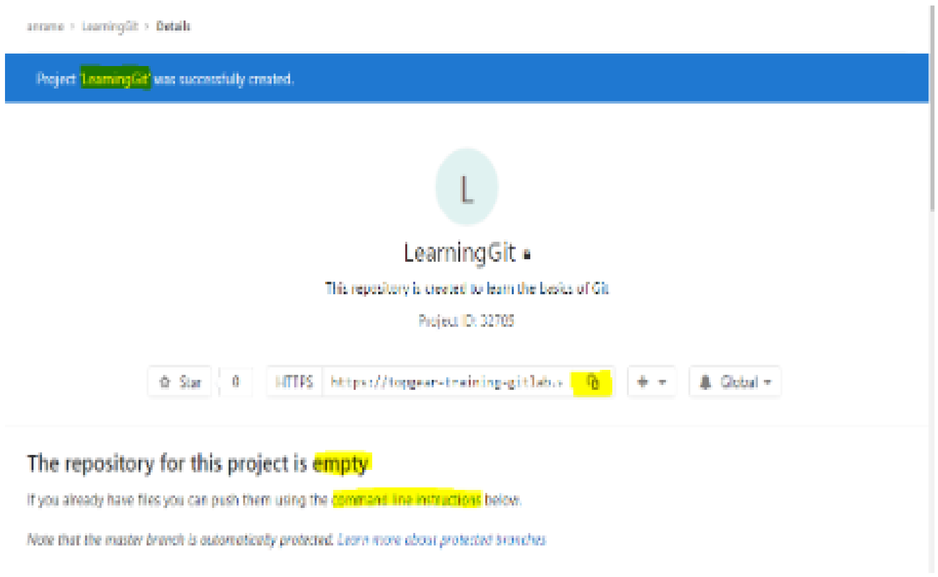
Visibility Level

- [ ] Private
  Project access must be granted explicitly to each user.
- [ ] Internal
  The project can be accessed by any logged in user.
- [✓] Public
  The project can be accessed without any authentication.

- [ ] **Initialize repository with a README**
  Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project   Cancel

Once you click on 'Create Project', the new repository will be created. The remote repository URL can be copied to the clipboard (highlighted in below pic)



anrame > LearningGit > Details

Project 'LearningGit' was successfully created.

LearningGit

This repository is created to learn the basics of Git

Project ID: 32705

The repository for this project is empty

If you already have files you can push them using the command line instructions below.

Note that the master branch is automatically protected. Learn more about protected branches

Push local repository details to Remote repository

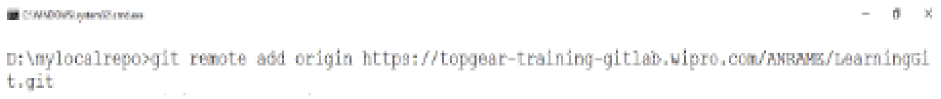
You can go to your working directory and issue the below commands to push your local repository to the remote repository

To add a new remote, use the git remote add command on the terminal, in the directory your repository is stored at.

<u>The git remote add command takes two arguments:</u>

A remote name, for example, "origin"

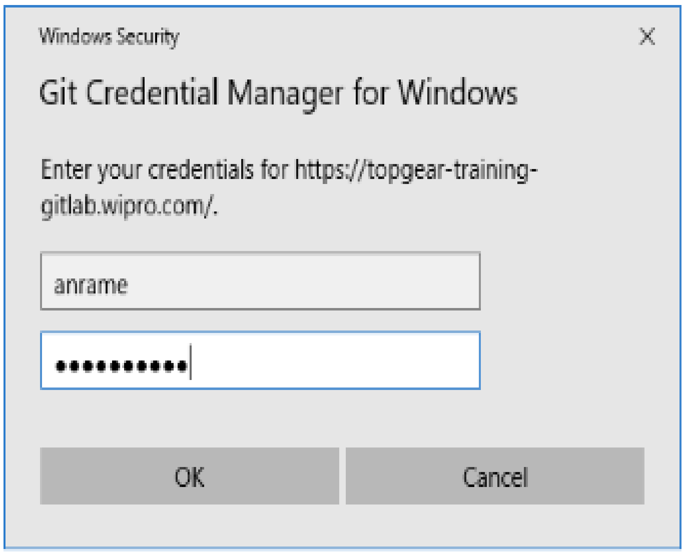
A remote URL, which you can find on the Source sub-tab of your Git repo



```
D:\mylocalrepo>git remote add origin https://topgear-training-gitlab.wipro.com/ANRAME/LearningGi
t.git
```

```
D:\mylocalrepo>git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 238 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://topgear-training-gitlab.wipro.com/ANRAME/LearningGit.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

The above command asks the user for his/her credentials and on providing, the same will push the files to the remote server



Windows Security

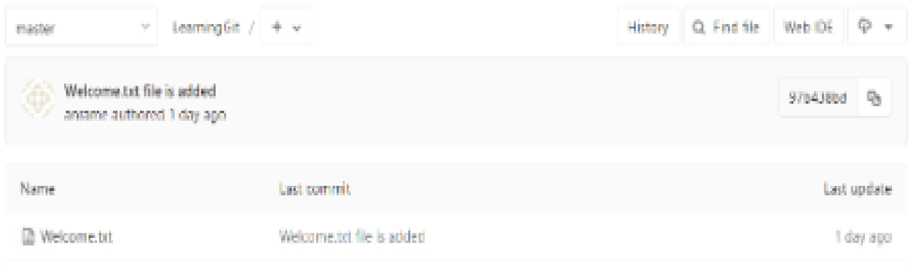
Git Credential Manager for Windows

Enter your credentials for https://topgear-training-gitlab.wipro.com/.

anrame

••••••••••

You can check the remote repository now to see if the files are updated in it



Welcome.txt file is added
anrame authored 1 day ago
979dJ8bd

Name	Last commit	Last update
Welcome.txt	Welcome.txt file is added	1 day ago

Download and integrate remote changes

If there are some new files present in the remote server, and you want your working directory be updated with that you need to use *git pull*

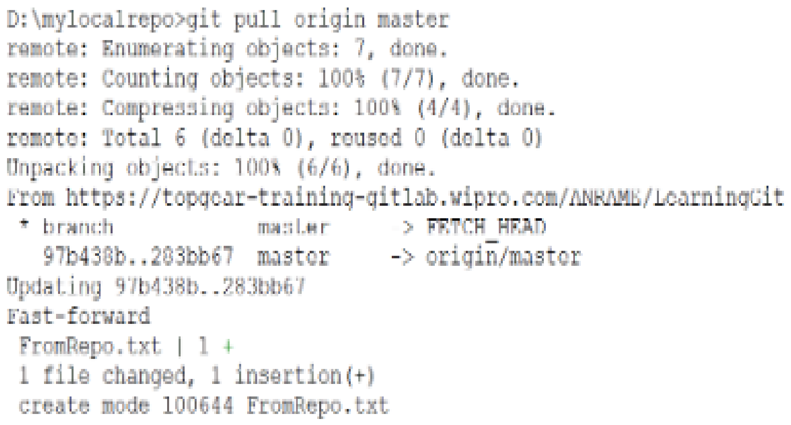
Create a new File called FromRepo.txt in the remote repository.

In your working directory issue the below commands

git checkout master

git pull origin master

```
D:\mylocalrepo>git checkout master
Already on 'master'
M       Welcome.txt
Your branch is up-to-date with 'origin/master'.
```

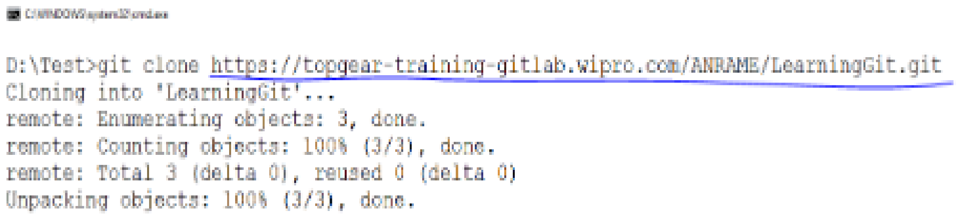


```
D:\mylocalrepo>git pull origin master
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
From https://topgear-training-gitlab.wipro.com/ANAME/LearningGit
 * branch            master     -> FETCH_HEAD
   97b438b..283bb67  master     -> origin/master
Updating 97b438b..283bb67
Fast-forward
 FromRepo.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 FromRepo.txt
```

These commands would help you have FromRepo.txt in your current working directory as well

Download an existing repository from a remote server

If there is already a remote repository available, and you want to create a clone of the repository, you can use *git clone*



```
C:\WINDOWS\system32\cmd.exe

D:\Test>git clone https://topgear-training-gitlab.wipro.com/ANRAME/LearningGit.git
Cloning into 'LearningGit'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
```

The "clone" command downloads an existing Git repository to your local computer. You will then have a full-blown, local version of that Git repo and can start working on the project.

Working with files in staging area

a) To remove a certain file from the staging area:

```
git    add    .
git    status
git    reset    head    filename_with_extension
```

b) To ignore files matching certain pattern from the staging area:

i) Open Notepad.
ii) Add the patterns to your file.
iii) Click "Save as" and select "all files".
iv) Save as **.gitignore**

Sample pattern to ignore .java files from all the folders: ***.java**

Sample pattern to ignore .txt files from all the folders: ***.txt**

To ignore a particular file from all the folders: ****/filename_with_extension**

Note: Your git repository should be aware of gitignore file in prior.

Working with files in local repository

a) To see history of a file (all the changes done in each commit):

```
gitk      filename_with_extension
```

b) To see history of the project (all the changes done in each commit):

```
gitk      projectName
```

View all the commits done so far with: `git      log       - - oneline`

c) To restore specific file(s) to a specific old version:

```
git      checkout      commit_id        - -        fileName_with_extension
git      checkout      commit_id        - -        file1      file2
```

d) To restore the entire project to a specific old version:

```
git      reset         - - hard        commit_id
```

e) To restore a deleted file from local repository:

```
git      add      .
git      status
git      restore        - - staged         filename_with_extension
git      restore        filename_with_extension
```

Pushing the project to remote repository

1. Create a new project in GitLab.
2. `git remote add origin URL_of_remote_repository`

   Example:

   ```
   git remote add origin https://topgear-training-gitlab.wipro.com/AB1234/MyProject.git
   ```
3. `git push - u origin master`

Getting the changes from remote repository and updating it in our local repository

1. **Two step pull from remote repository (Fetch and Merge):**

   Fetch: `git fetch`

   View the new changes we had received: `git diff master origin/master`

   Update the changes to your local working copy: `git merge`

2. **One step pull from remote repository:** fetches the new commits and merges these into your local working copy.

   ```
   git checkout master
   git pull origin master
   ```

Branching and Merging

Branch

- In Git, branches are a part of your everyday development process.
- Git branches are effectively 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 how small—you spawn a new branch to encapsulate your changes.
- A branch represents an independent line of development.
- Branches serve as an abstraction for the edit/stage/commit process.
- You can think of them as a way to request a brand new working directory, staging area, and project history.

Merging

- Merging is Git's way of putting a forked history back together again.
- It lets you take the independent lines of development created by branching and integrate them into a single branch.

Let us assume this scenario:

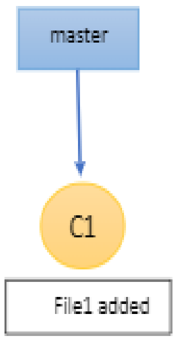
1. You have a **master** branch where couple of commits are done.
2. A) You create a new branch **NewBranch_1** for adding some new features and changes.

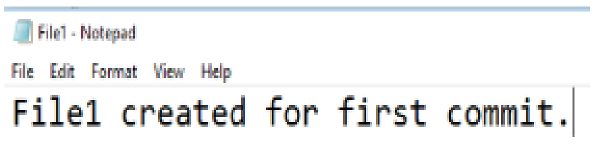
   B) You started working on the **NewBranch_1**.

   C) You have committed the changes to **NewBranch_1** but it is not yet merged into master.

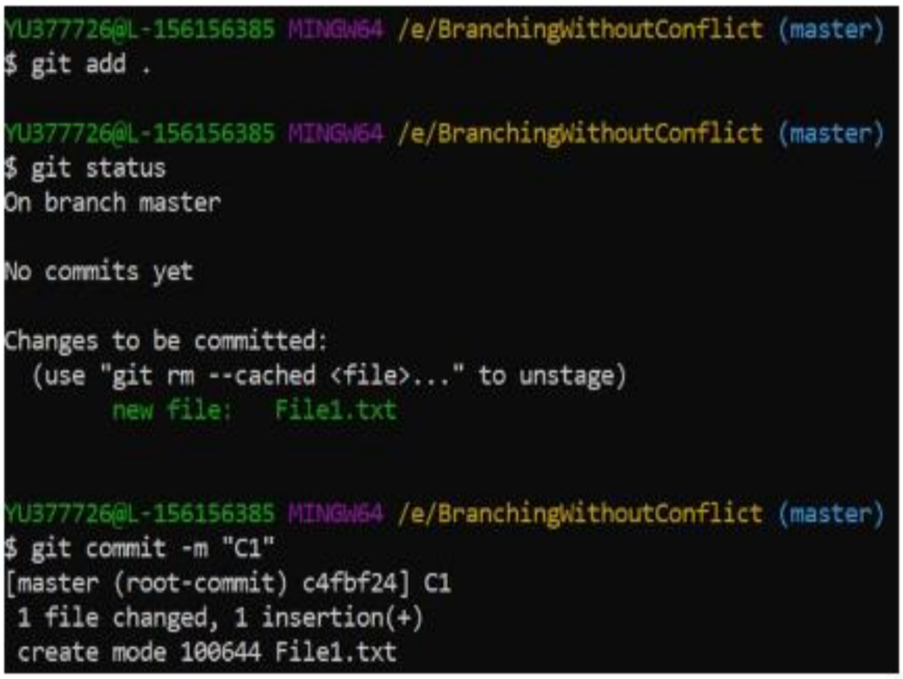
3. Your lead is asking you to fix another issue immediately, so you switch back to **master** and create a new branch **NewBranch_2**.
4. You had fixed the issue, tested and merged it into **master** branch.
5. You switch back to **NewBranch_1** for testing the changes you have done earlier.
6. You are trying to merge **NewBranch_1** into **master**.

Step 1: You have a **master** branch where couple of commits are done.

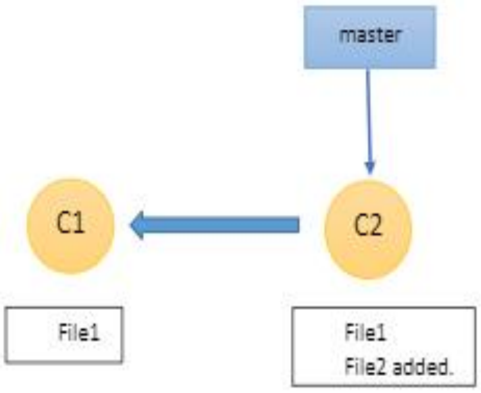


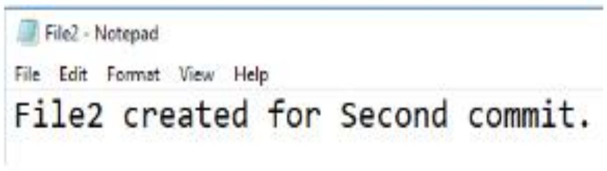


```
File1 created for first commit.
```

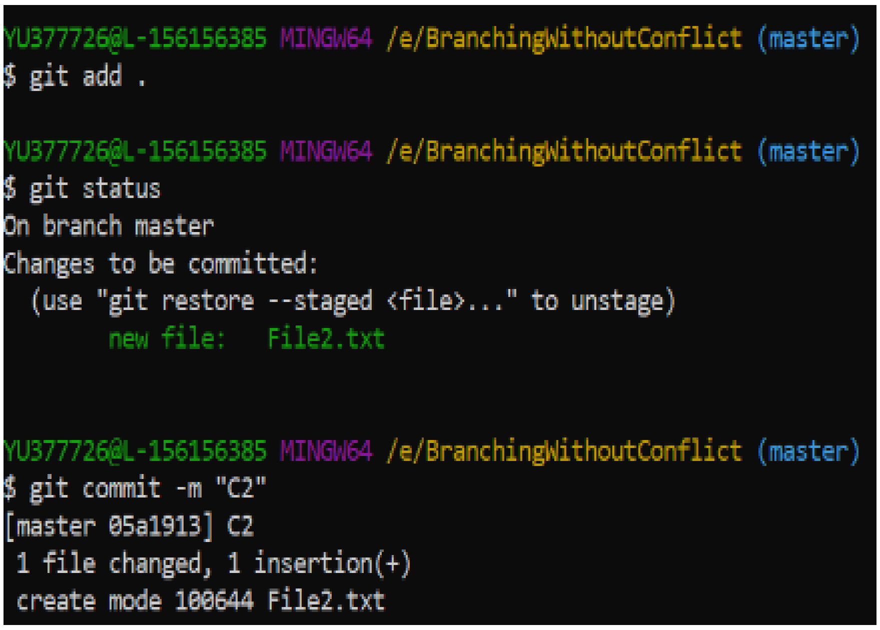


```
YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git add .

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   File1.txt


YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git commit -m "C1"
[master (root-commit) c4fbf24] C1
 1 file changed, 1 insertion(+)
 create mode 100644 File1.txt
```

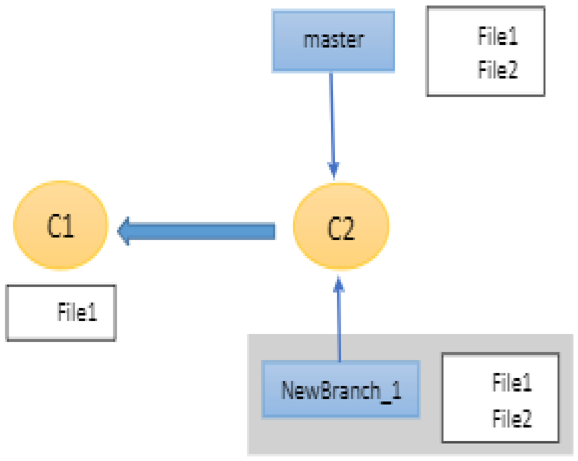


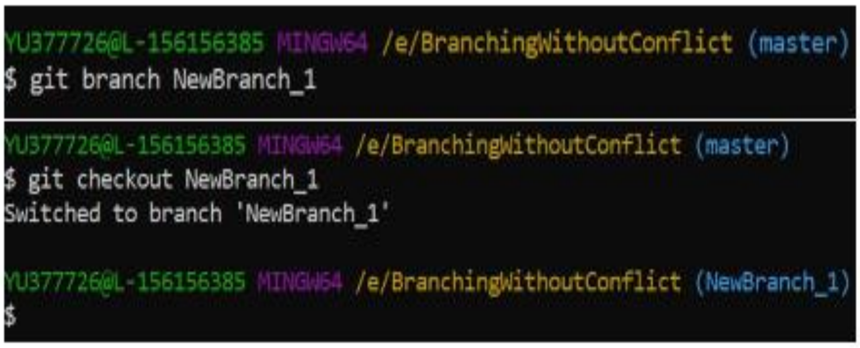


File2 created for Second commit.


```
YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git add .

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   File2.txt


YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git commit -m "C2"
[master 05a1913] C2
 1 file changed, 1 insertion(+)
 create mode 100644 File2.txt
```

Step 2: A) You create a new branch **NewBranch_1** for adding some new features and changes.

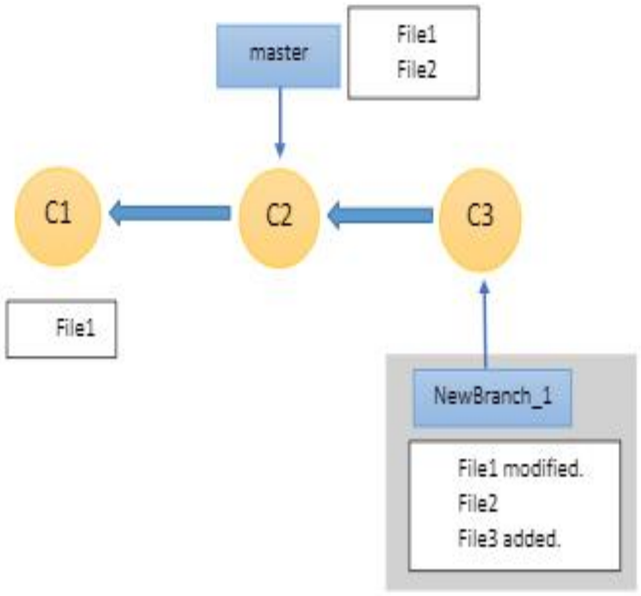


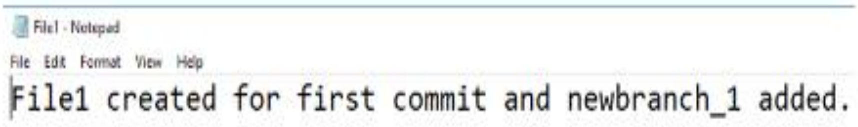


```
YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git branch NewBranch_1
```

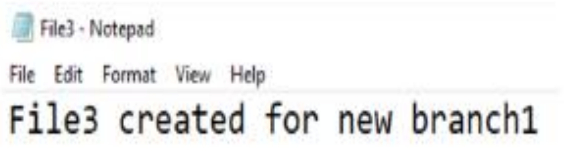
```
YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git checkout NewBranch_1
Switched to branch 'NewBranch_1'

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_1)
$
```

Step 2: B) You started working on the NewBranch_1.



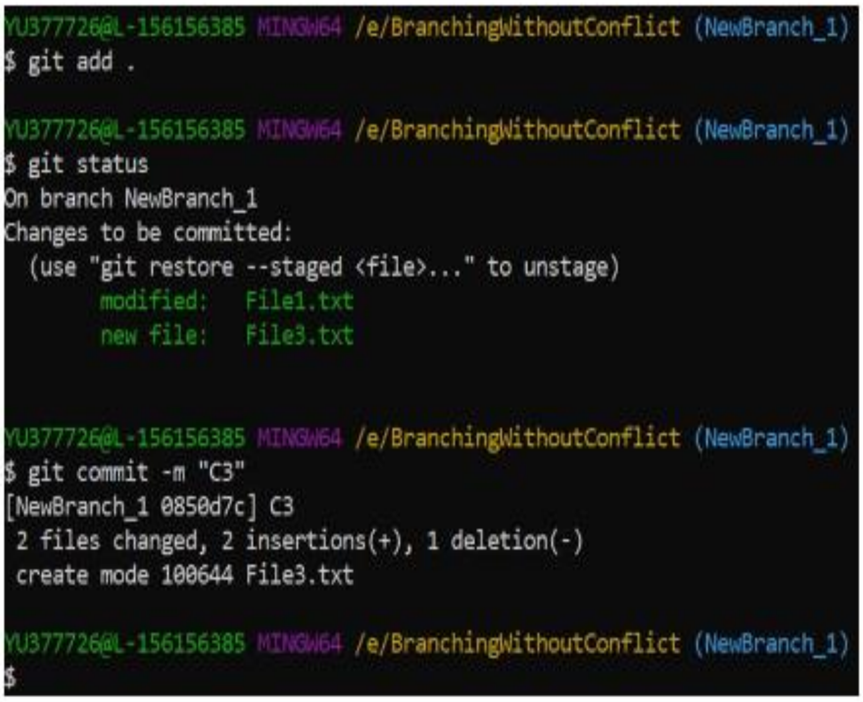


File1 created for first commit and newbranch_1 added.



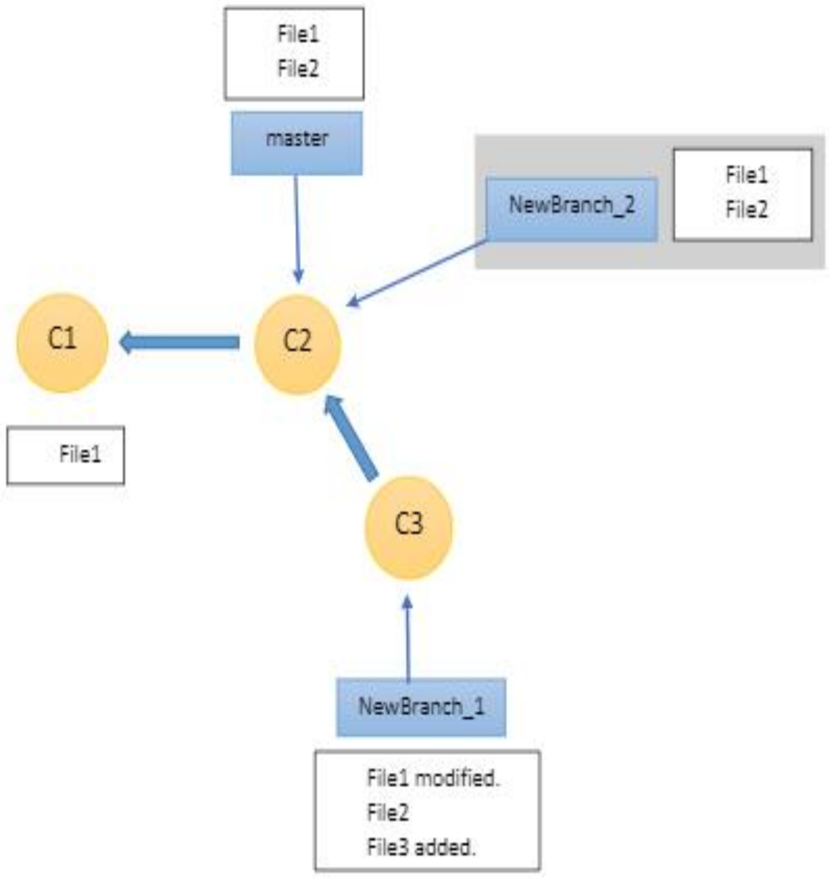
File3 created for new branch1

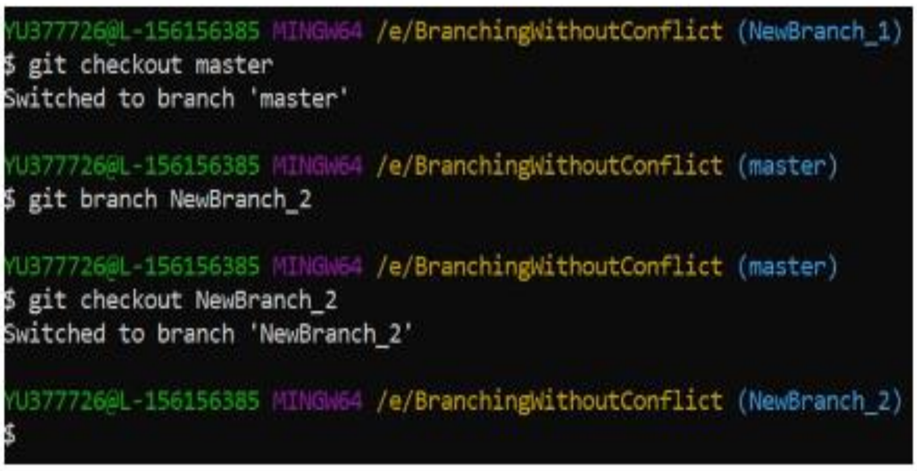
Step 2: C) You have committed the changes to **NewBranch_1** but it is not yet merged into **master**.



```
YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_1)
$ git add .

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_1)
$ git status
On branch NewBranch_1
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   File1.txt
        new file:   File3.txt


YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_1)
$ git commit -m "C3"
[NewBranch_1 0850d7c] C3
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 File3.txt

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_1)
$
```

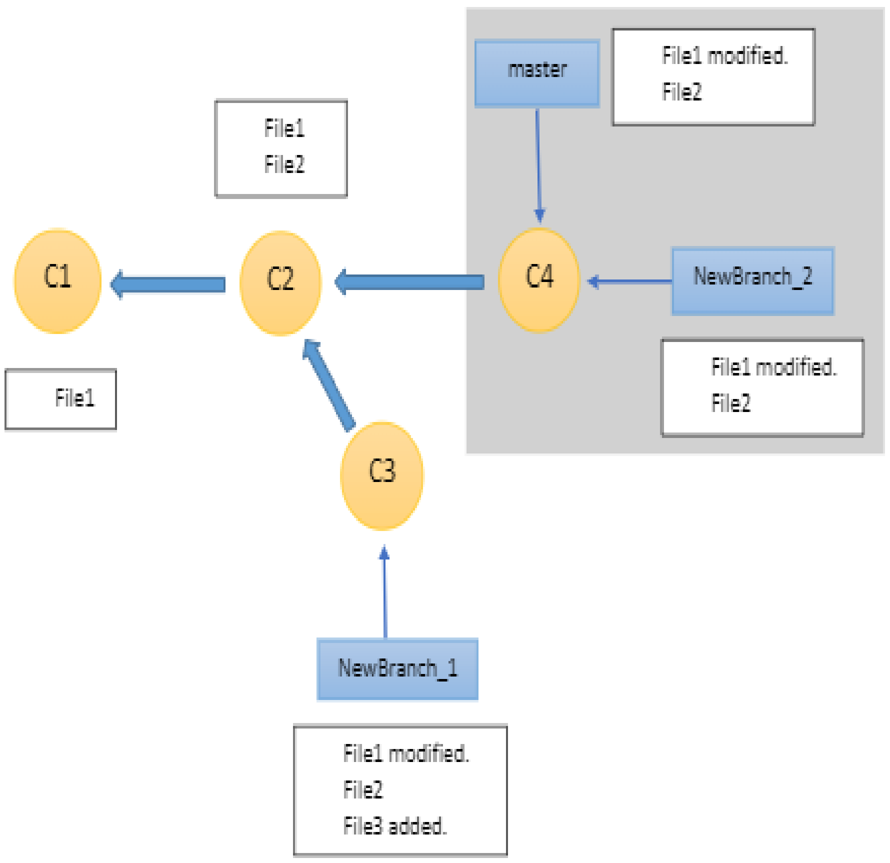
Step 3: Your lead is asking you to fix another issue immediately, so you switch back to **master** and create a new branch **NewBranch_2**.

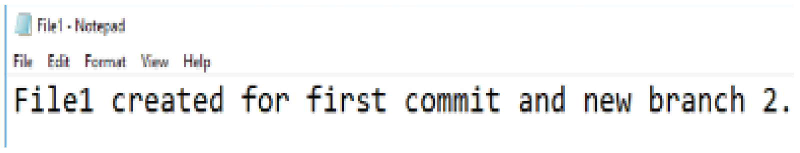




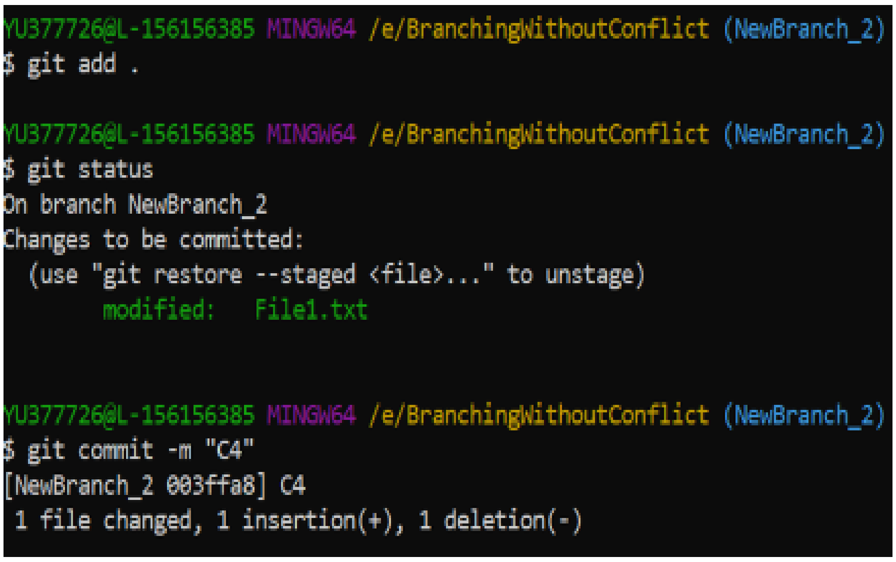
```
YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_1)
$ git checkout master
Switched to branch 'master'

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git branch NewBranch_2

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git checkout NewBranch_2
Switched to branch 'NewBranch_2'

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_2)
$
```

Step 4: You had fixed the issue, tested, committed and merged it into **master** branch.

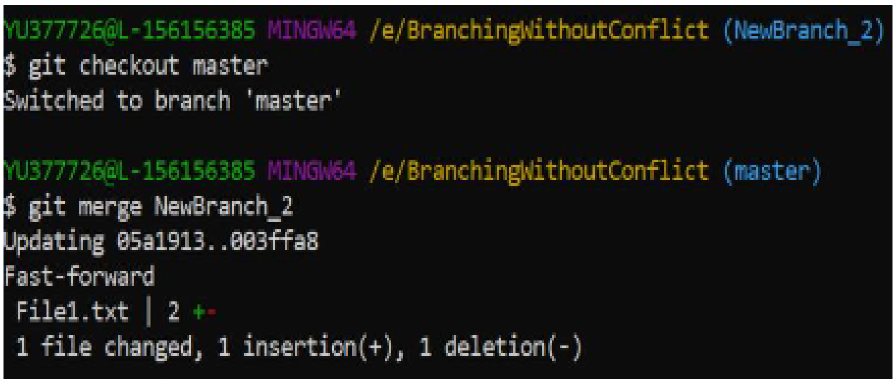




```
File1 created for first commit and new branch 2.
```



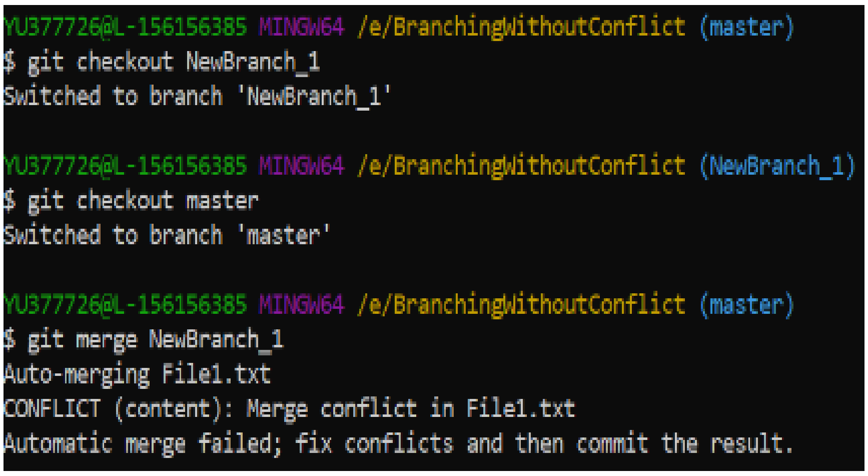
```
YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_2)
$ git add .

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_2)
$ git status
On branch NewBranch_2
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   File1.txt


YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_2)
$ git commit -m "C4"
[NewBranch_2 003ffa8] C4
 1 file changed, 1 insertion(+), 1 deletion(-)
```



```
YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_2)
$ git checkout master
Switched to branch 'master'

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git merge NewBranch_2
Updating 05a1913..003ffa8
Fast-forward
 File1.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

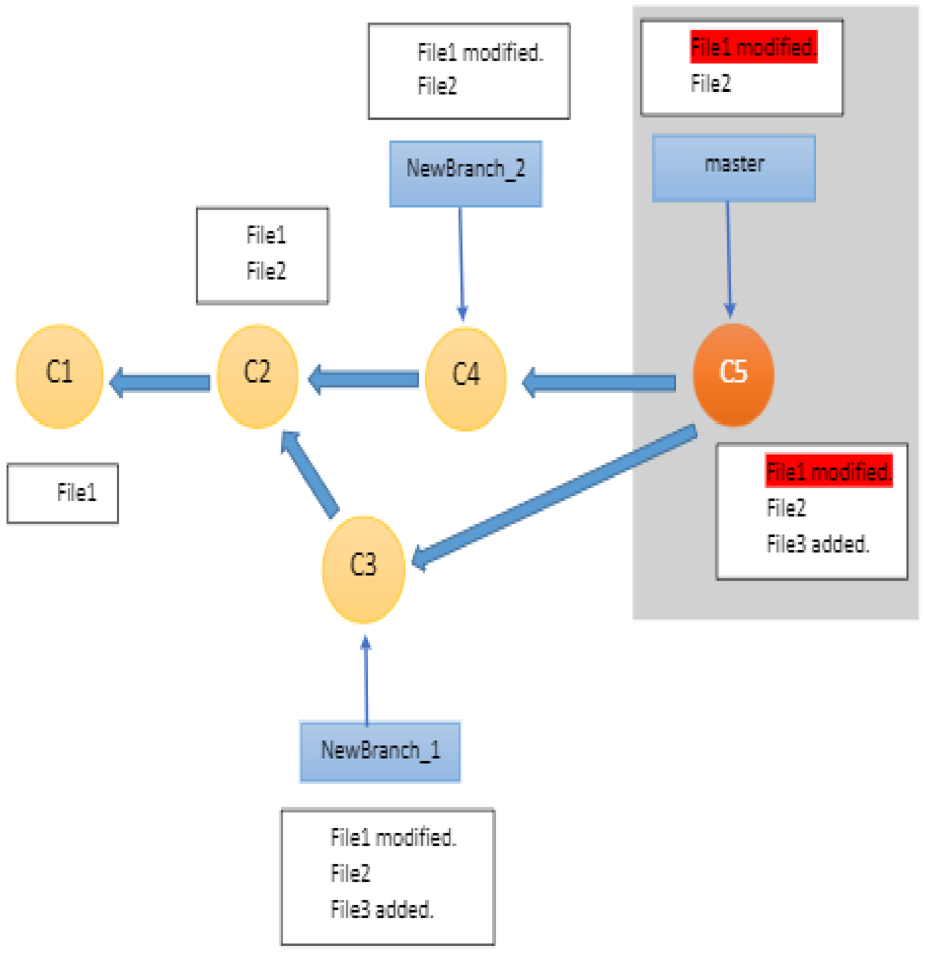
Step 5: You switch back to NewBranch_1 for testing the changes you have done earlier.

Step 6: After testing, switch back to master and try merging NewBranch_1 into master.

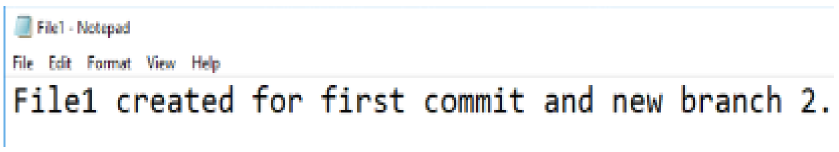


```
YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git checkout NewBranch_1
Switched to branch 'NewBranch_1'

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (NewBranch_1)
$ git checkout master
Switched to branch 'master'

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master)
$ git merge NewBranch_1
Auto-merging File1.txt
CONFLICT (content): Merge conflict in File1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Merge conflicts

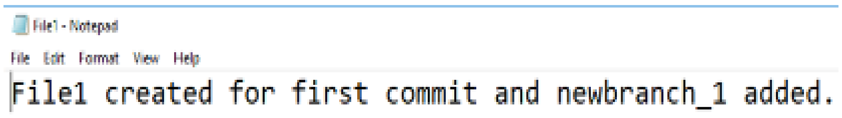


Changes done to File1 in NewBranch_2:



```
File1 created for first commit and new branch 2.
```

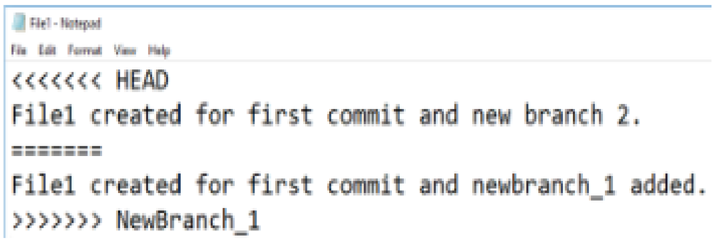
Changes done to same File1 in NewBranch_1:



```
File1 created for first commit and newbranch_1 added.
```

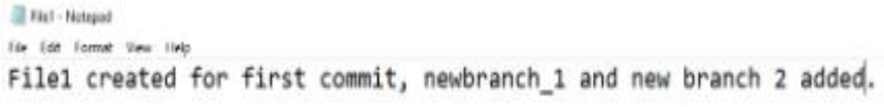
You had changed the same part of the same file differently in the two branches you are merging. Git won't be able to merge them cleanly.

Now your File1 contains standard conflict-resolution markers:



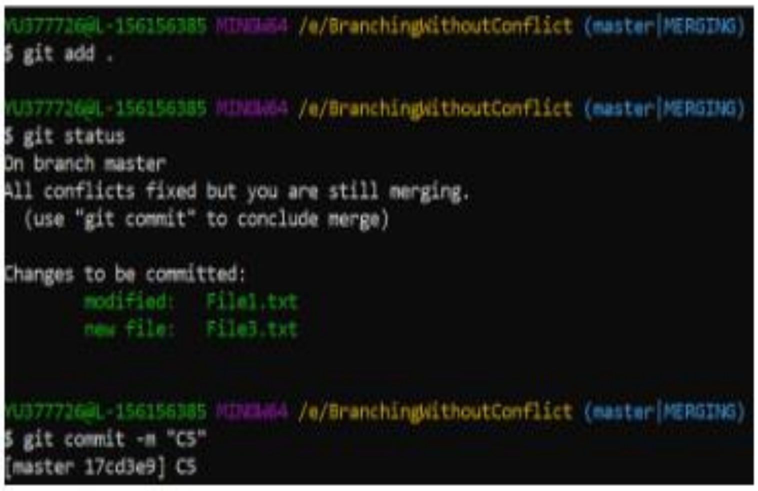
```
<<<<<<< HEAD
File1 created for first commit and new branch 2.
=======
File1 created for first commit and newbranch_1 added.
>>>>>>> NewBranch_1
```

This means the version in HEAD (your master branch, because that was what you had checked out when you ran your merge command) is the top part of that block (everything above the =======), while the version in your NewBranch_1 branch looks like everything in the bottom part. In order to resolve the conflict, you have to either choose one side or the other or merge the contents yourself.



```
File1 created for first commit, newbranch_1 and new branch 2 added.
```

This resolution has a little of each section, and the <<<<<<<, =======, and >>>>>>> lines have been completely removed. After you have resolved each of these sections in each conflicted file, run git add on each file to mark it as resolved. Staging the file marks it as resolved in Git.



```
YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master|MERGING)
$ git add .

YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master|MERGING)
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
        modified:   File1.txt
        new file:   File3.txt


YU377726@L-156156385 MINGW64 /e/BranchingWithoutConflict (master|MERGING)
$ git commit -m "C5"
[master 17cd3e9] C5
```