**SWE 525 Git Version Control Assignment 1 8/10/16**

**ZAMEER AHMED (90641)**

A. Answer following questions briefly:

1. List out the key difference between a centralized version control system and distributed version control system.

**Sol:** One principal tool in any product task is a Version Control System (VCS). A VCS helps you monitor highlights, bug tickets, and some other changes in your task. It is additionally extremely help-ful to keep went down forms of your work just in the event that anything turns out badly. There are a considerable amount of instruments accessible to address this issue, including: Perforce, Mercurial, SVN, Git, CVS, and so forth and they can be arranged into two classes: Centralized and Distributed.

In Centralized Version Control Systems (CVCS), we keep up a focal vault that is shared among all colleagues. Every colleague makes a neighborhood store, rolls out improvements to their nearby duplicate and confer their adjustments to the common storehouse, to be seen by whatever is left of the group.

In Distributed Version Control Systems (DVCS), we keep up an expert archive which is controlled by project administration affirmed watchmen or directors. From this expert repo, every colleague clones the expert store into their nearby vault, roll out improvements and focus on their cloned form and after that push his progressions back to the primary shared archive. The watchmen at the expert storehouse choose whether to acknowledge these confers or reject them.

1. List down any two centralized version control system and 2 distributed version control system

**Sol:** Distributed VCS: Git, Mercurial.

Centralized VCS: CVS, Perforce, SVN.

1. What are the advantages of git VCS over other VCS

**Sol:**

* It is straightforward to use.
* It bolsters non-direct advancement.
* It can deal with vast activities.
* Stretching and combining is simple.

1. What are the different states of a file in the Git VCS

**Sol:**

* Untracked: The document is not followed by the Git archive. This implies the document was neither staged nor committed.
* Tracked: Committed but not staged.
* Staged: Organized to be incorporated into the following commit
* Modified: The document has changed yet the change is not staged

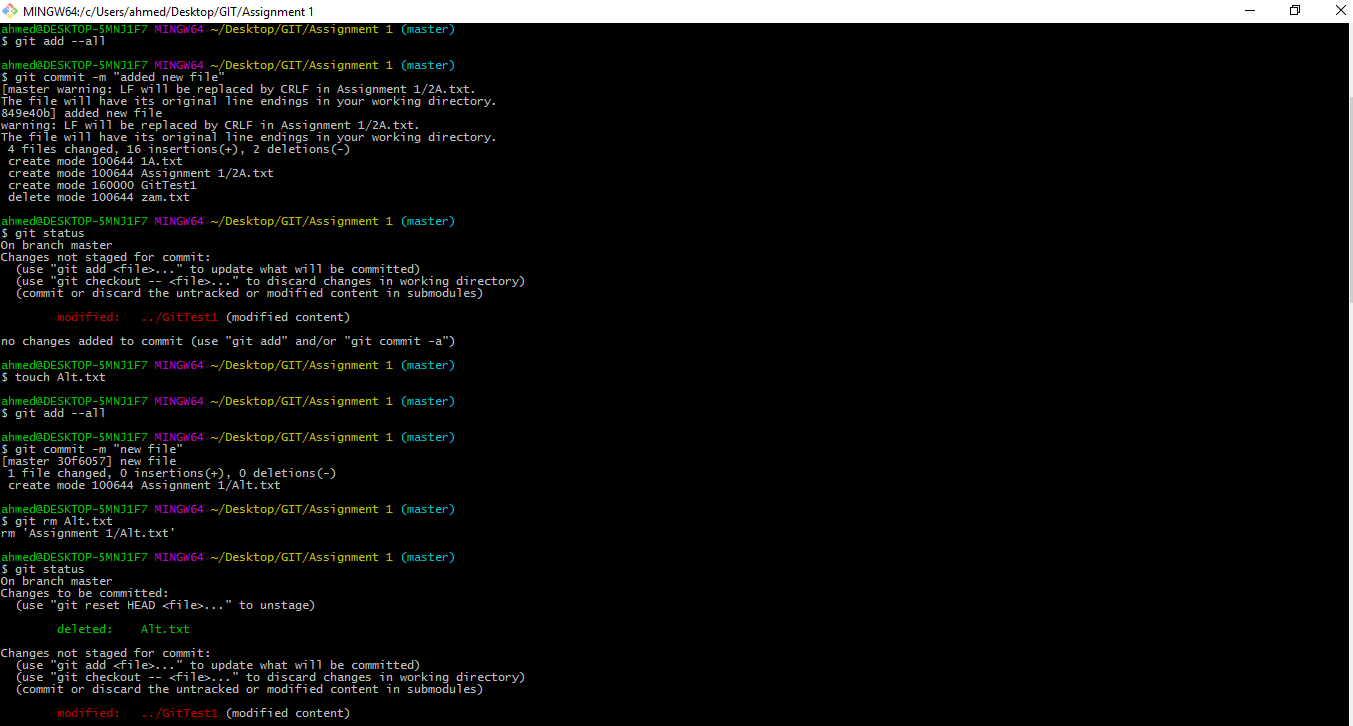
**B. GIT REMOTE REPOSITORIES:** Perform following tasks and explain how you performed each operation. Draw a flow diagram as you progress through the steps. Add all git commands you used and push the repository in your github. Add your github public repository (for these following tasks) link with the homework.

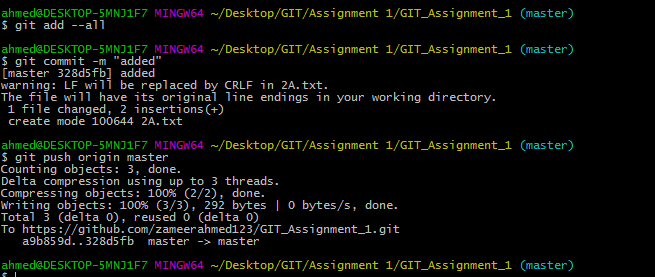
1. Clone an existing repository on Github created during course and configure your local repo to point to the remote repository

* Creating a repo01 directory.
* Initializing git repository for the current directory repo01. By, entering the command “git init”. But before that we have to check whether we are in the working directory that is repo01. If we are not in the present working directory then enter command “cd repo01”.
* Created 3 files File1.txt and File2.txt and File3 in MyRepo directory using touch command.
* Now enter command git status. We get list of files which we added but not committed yet. So these files are called as untracked files.
* Add this files to the commit “git add - -all”. It commits all untracked files at once.
* Now commit and name that commit by entering command; “git commit -m “first commit”.
* Push to github using the URL of the repository created using command “git push —set-upstream origin master”

1. Perform some operation like add, remove, modify and finally push your changes to the remote repository

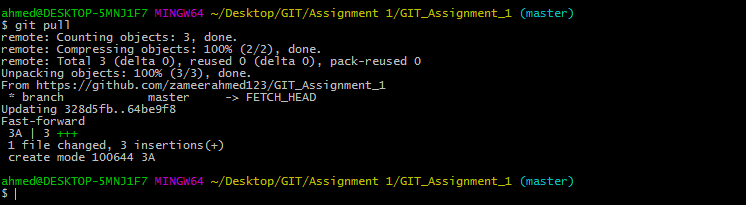
* Added new file modified and pushed it to remote Repository. Below are changes



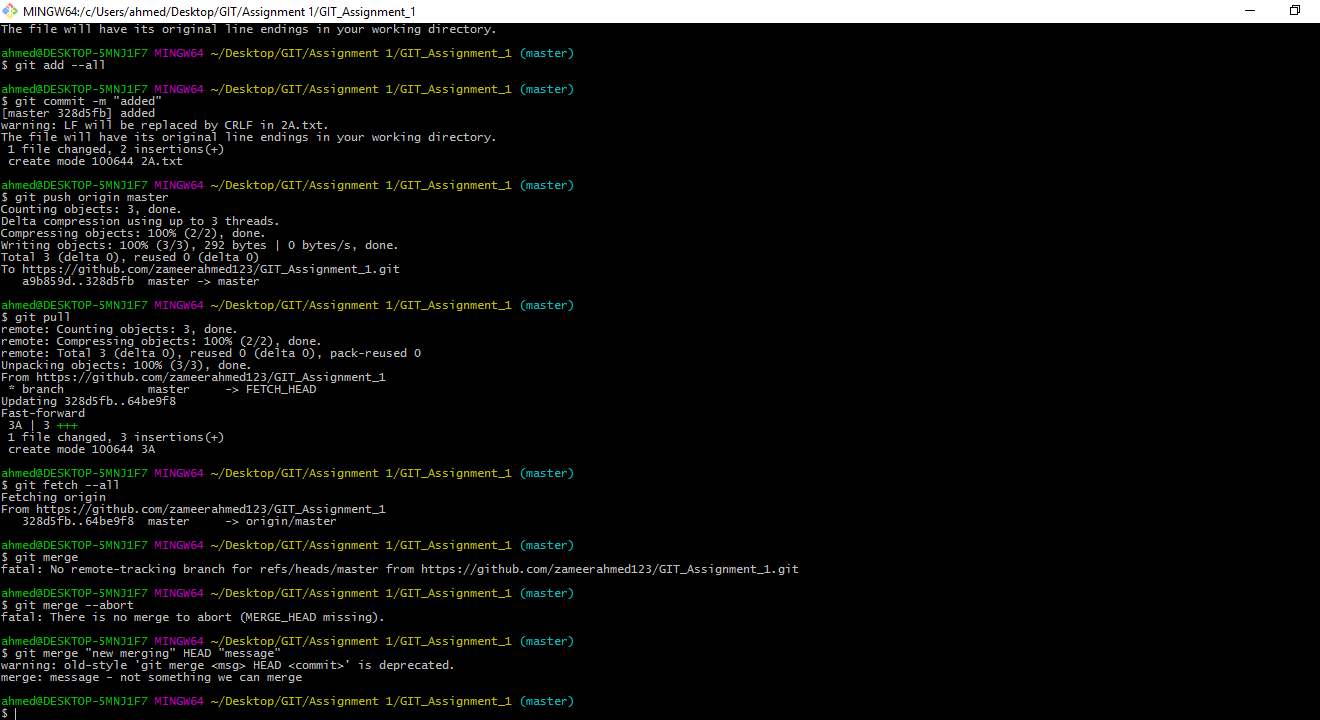


1. Pull the latest changes from the repository to get the updates from others in to your local repo and merge the changes

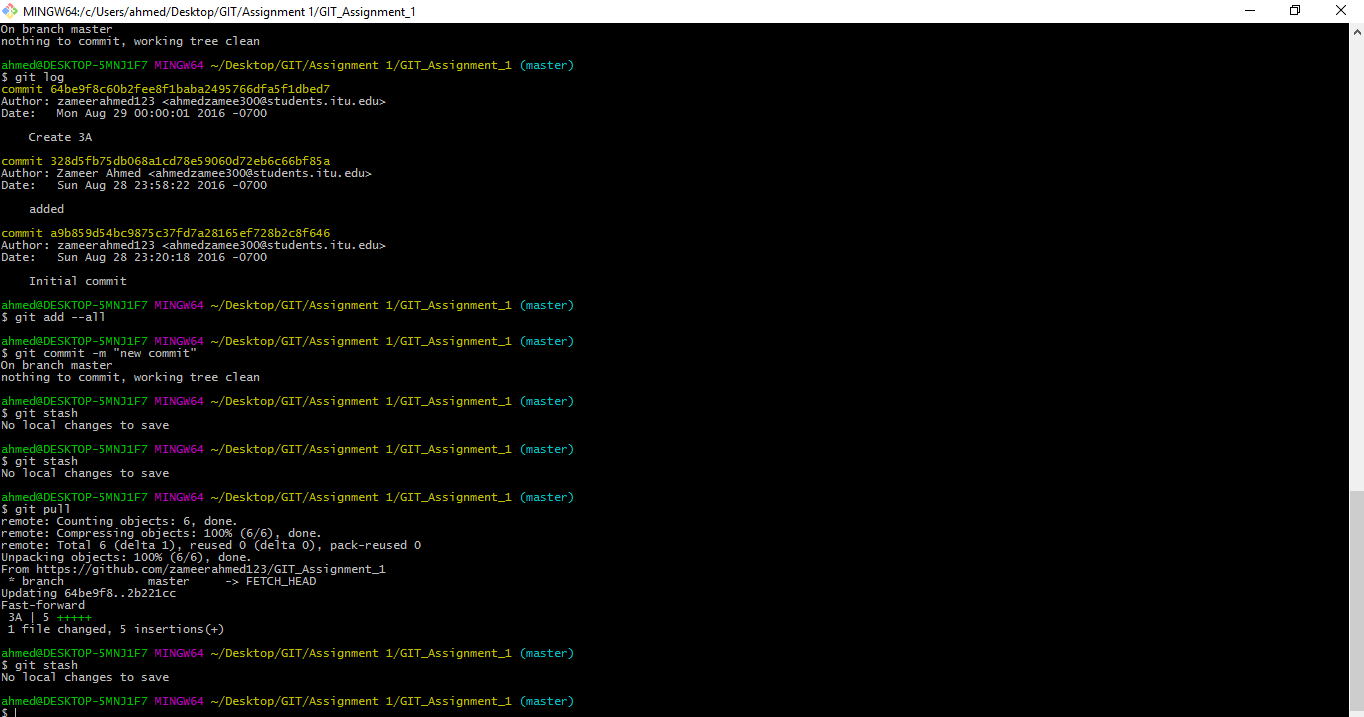
* Used git pull command to pull the new and modified file to the local



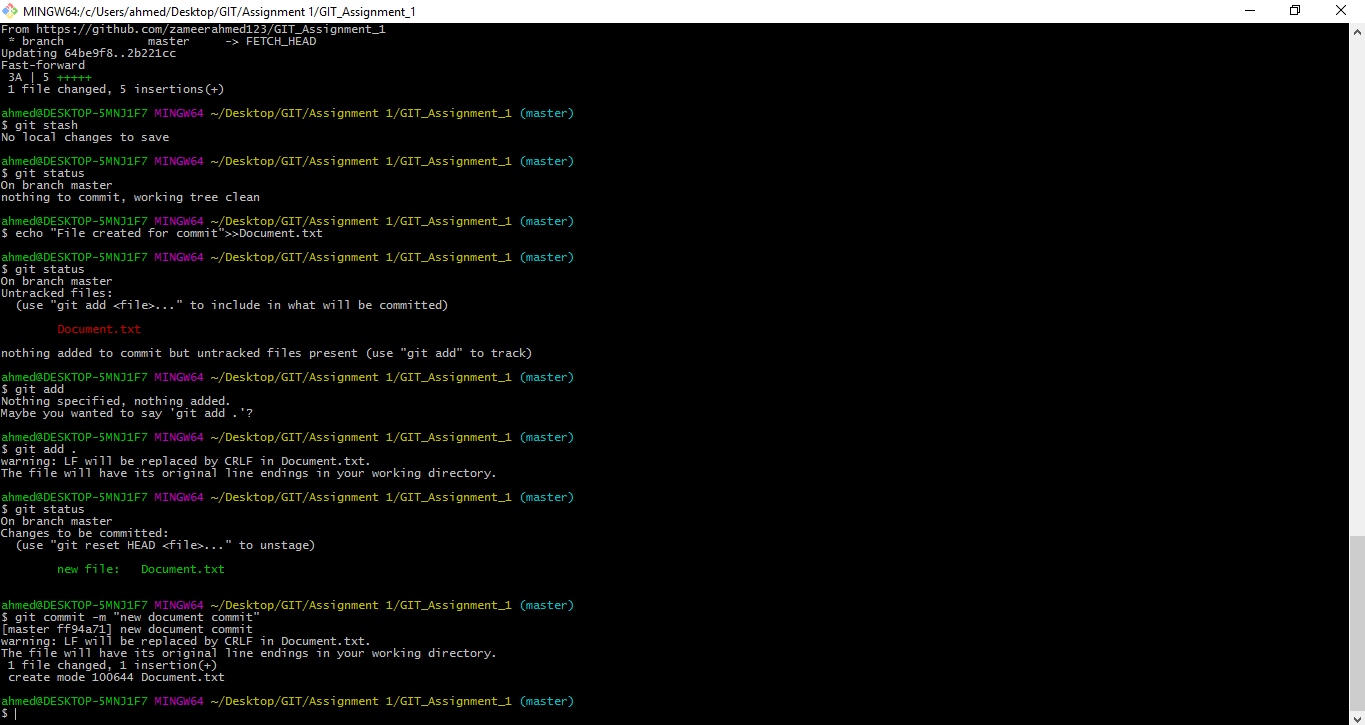
1. Try fetching the changes and perform the merge to get the difference between the pull and the merge command

1. Perform some changes and before committing the changes, stash your changes and then pull the changes and finally apply the changes to understand how stashing works

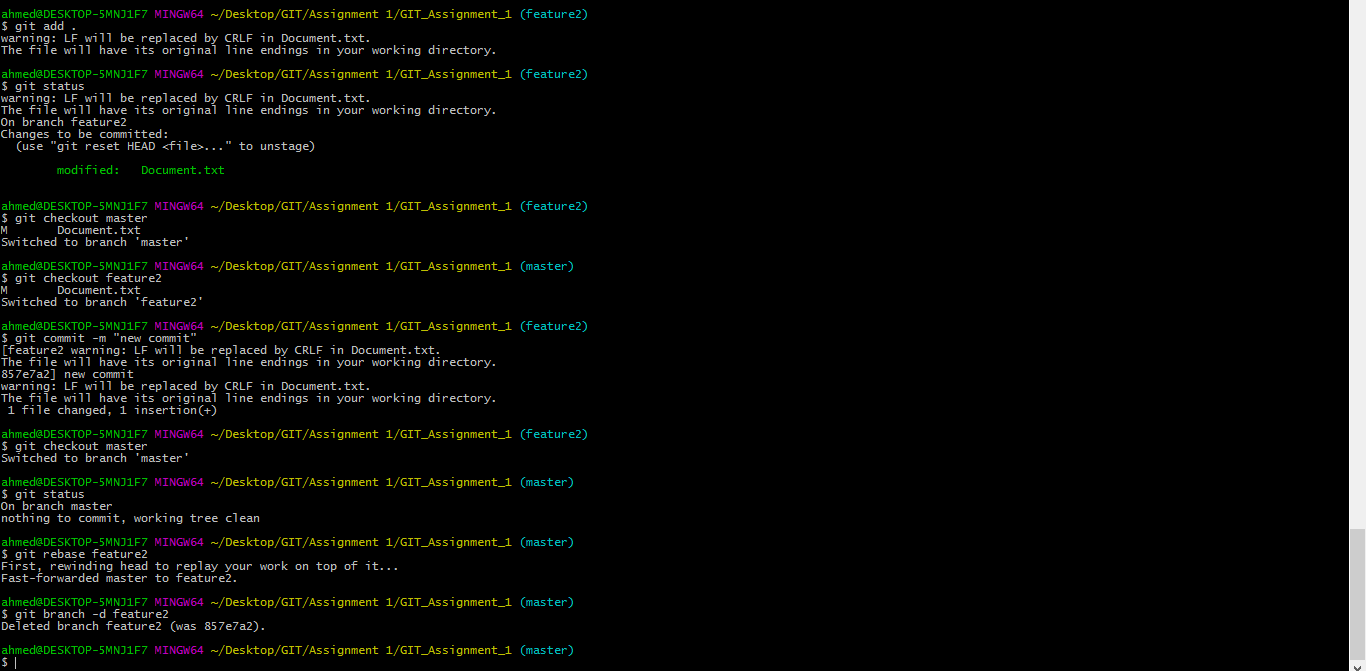


1. Create a feature branch and do some file operations in the branch and commit the changes to the branch



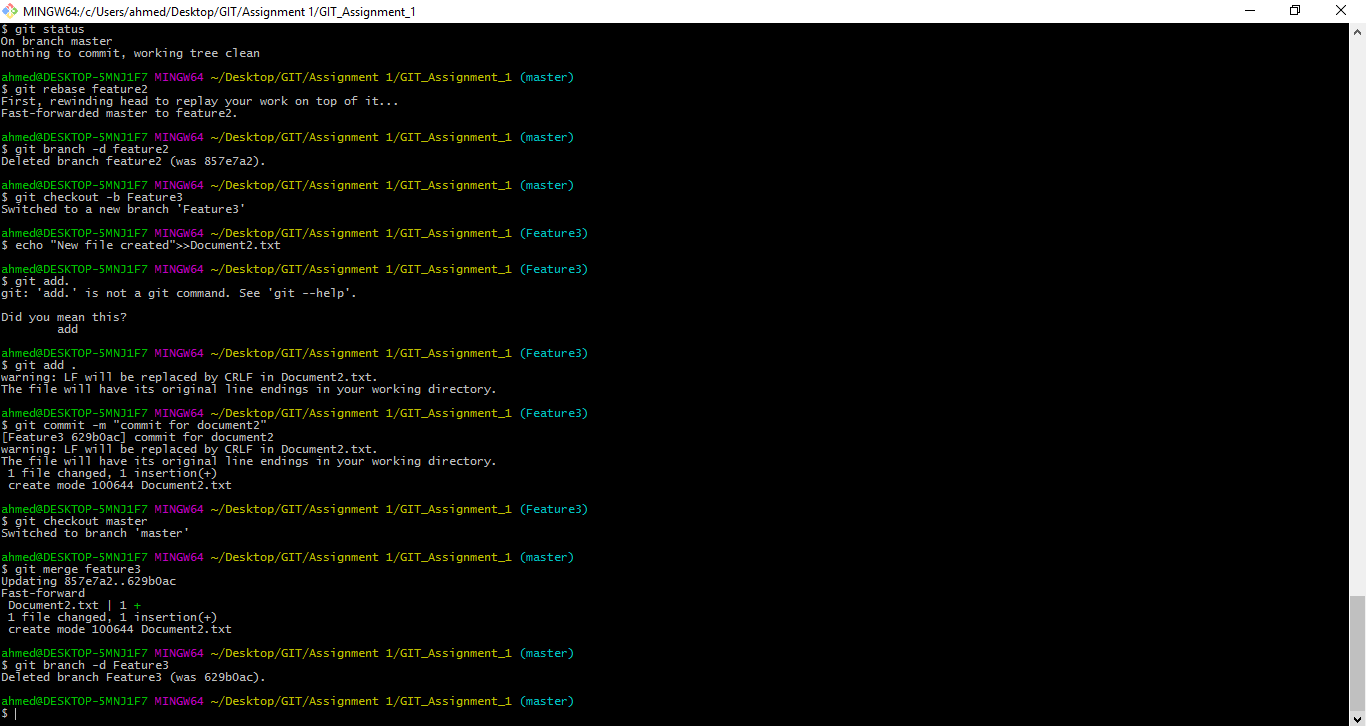
1. Merge the changes using the rebase command and finally perform a safe deletion of the feature branch

* **git rebase** to merge the branches
* **git branch –d feature** to delete the feature2 branch



1. Create another feature branch and this time after committing the changes to the feature branch, merge the changes using fast forward merge and then delete the feature branch

* Git checkout –b feature3
* Git add .
* Git commit –m “”
* Git checkout master
* Git merge feature3
* Git branch –d feature3



**Work Flow:**

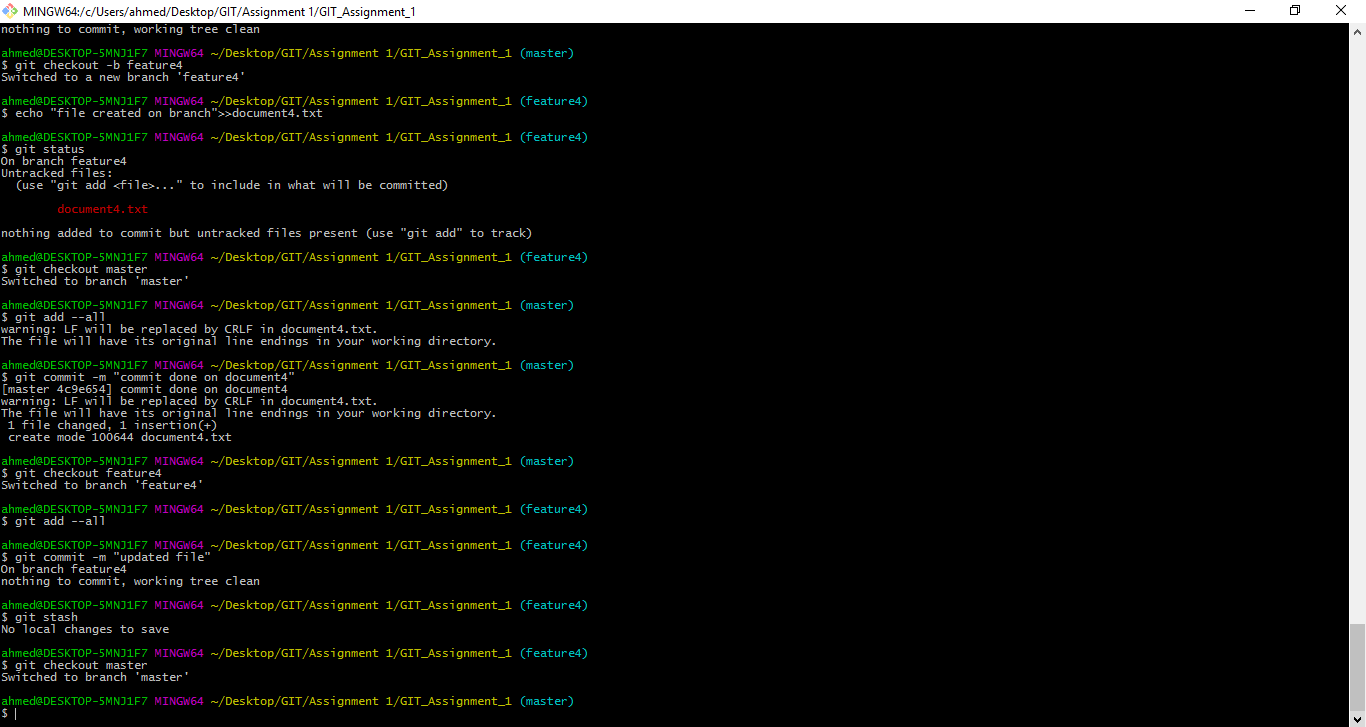
* Git add --all 🡪git commit - m 🡪 git pull 🡪 git stash 🡪 git checkout master 🡪 git rebase feature2 🡪 gitbranch -d feature2 🡪 git checkout -b feature3 🡪 echo “new file document.txt” 🡪 git add --all 🡪 git commit –m 🡪 git checkout master 🡪 git merge feature3 🡪 git branch –d feature3

**C. GIT BRANCHING AND MERGING:** Perform following tasks and explain how you performed each operation. Draw a flow diagram as you progress through the steps. Add all git commands you used and push the repository in your github. Add your github public repository (for these following tasks) link with the homework.

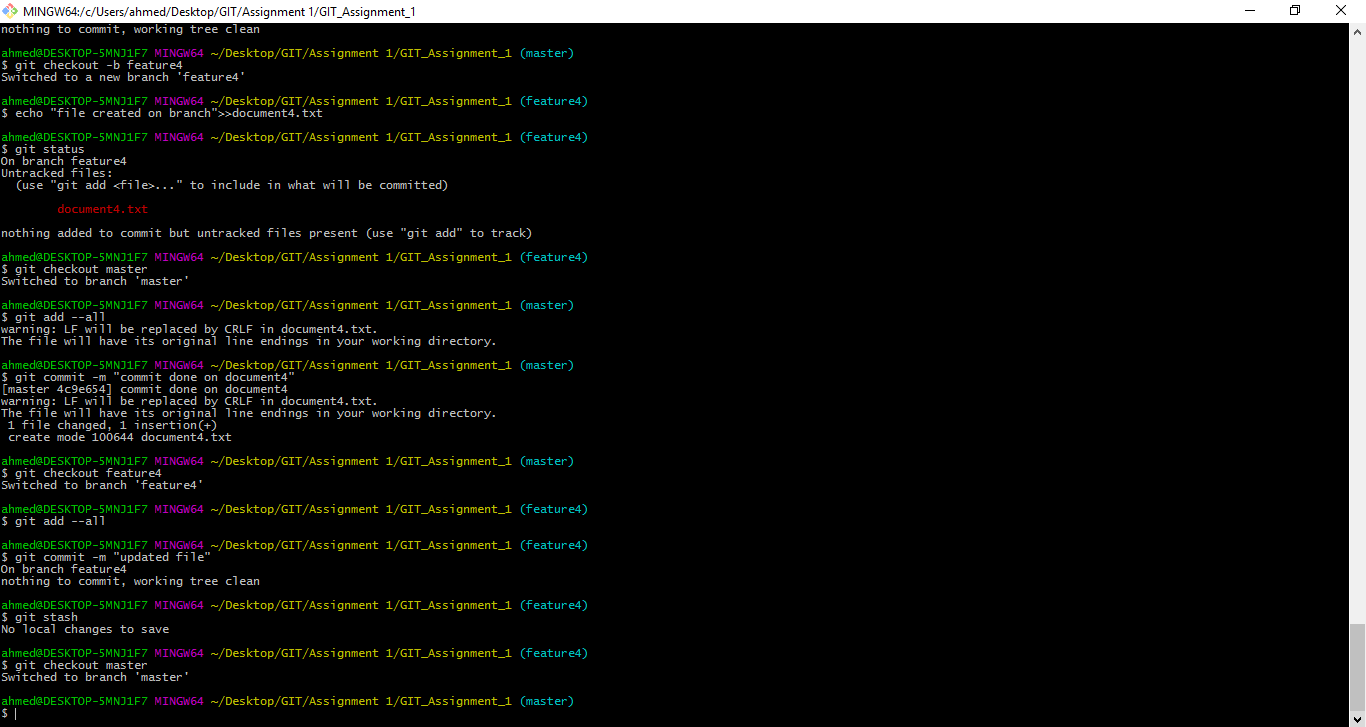
1. Create a local branch using git checkout -b branchname command

* Git checkout –b feature4

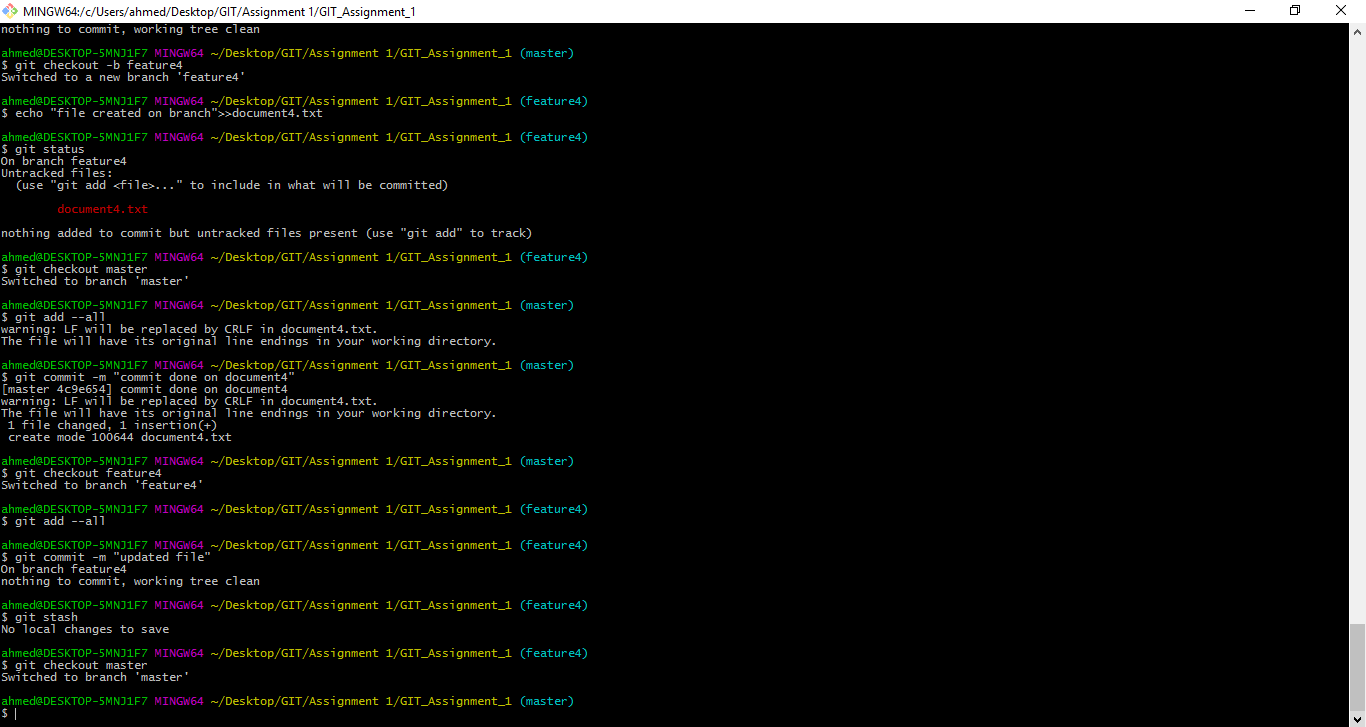
1. Observe the difference by doing some file operations and switch back to the master branch and see if you can see the changes done on the branch



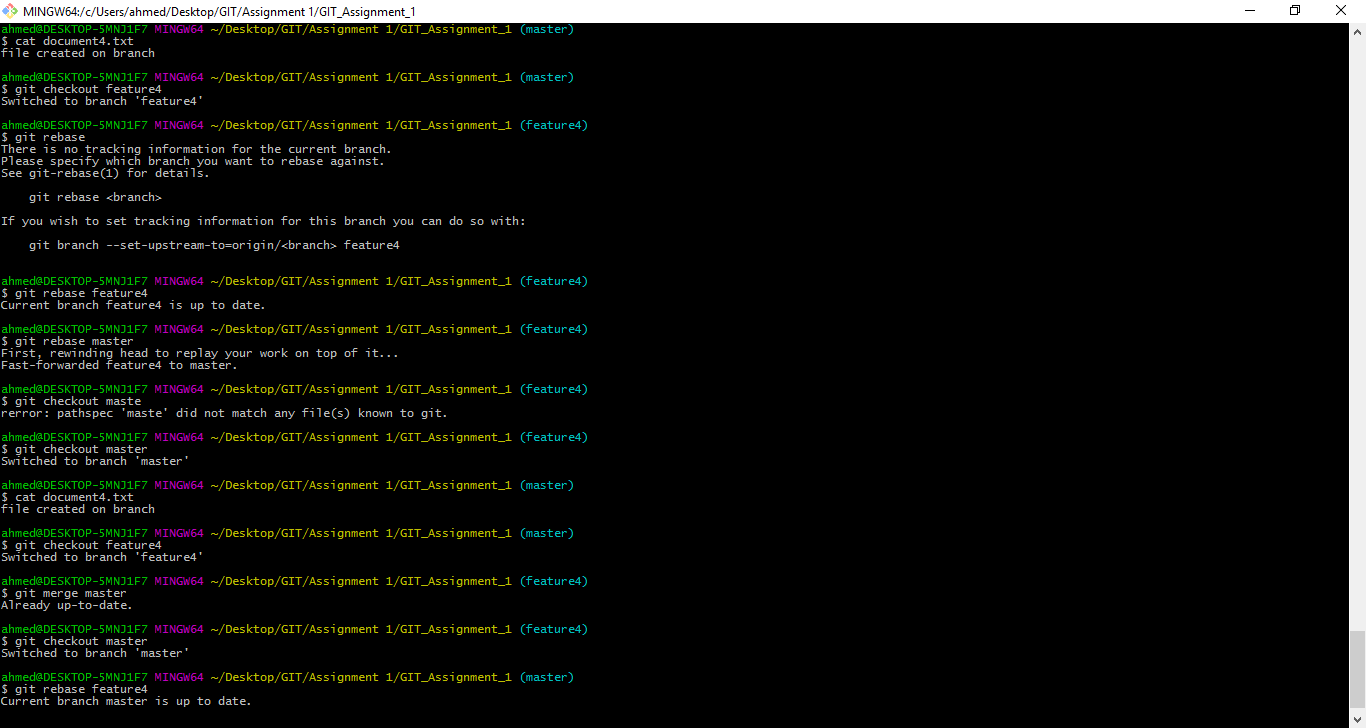
1. Now switch back to the branch and commit the changes and switch to master branch. Now see if you can still see the changes in the master branch



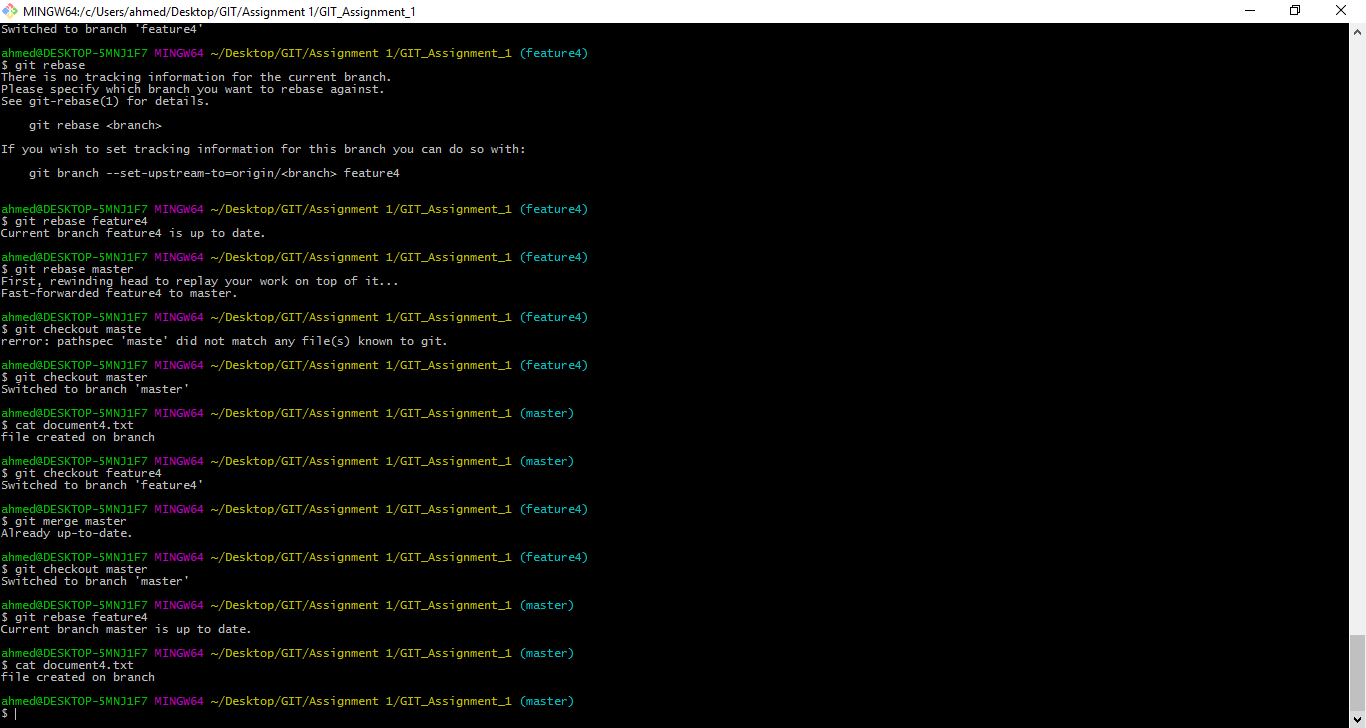
1. Now switch back to the branch name and stash the changes and apply the changes to the master branch by switching to the master branch



1. Try merging the changes from the branch to the master branch using all the three merge strategies and then view the git log



1. Push the local branch to the remote repository and see if the branch is present on the remote repository – Github



**D: Review following “MERGE with CONFLICT” scenario. Complete the exercise as requested below:**

**Purpose:**

Learn how to merge when there are code conflicts

Learn how to interact with the remote repository

**Preparation**

**Watch following video:** [**https://vimeo.com/138418055**](https://vimeo.com/138418055)

**You** may want to download the script used for video**. (copied at the end of this assignment)**

**Exercise**

**For t**his exercise, you should experiment with the merging files that have conflicts. You may use the code below or an*y* code of your choice.

public class TheMotivator {

public void feedback(int score) {

if (score == 100)

System.out.println("You're awesome");

else if (score > 90)

System.out.println("That's great");

else if (score > 60)

System.out.println("That's good ");

else

System.out.println("Well, what can I say?");

}

public static void main(String[] args) {

TheMotivator tm = new TheMotivator();

tm.feedback(60);

}

}

**Specific Requirements**

Turn in a git log that contains at least two different branches that have been merged. NOTE: The point of this is to get comfortable with git, so you do not need to follow the demo exactly. The log should show some reasonable amount of branching and merging. You should do some merges with and without conflicts, although we won't be able to tell this from the log. Show with diagram also.

Remember that to create log file you do:

git log --pretty=format:"%s" --graph > mylog.txt

I would suggest you spend at least half an hour on this task (more if this is all new to you).

**Submit**

Explain steps followed, draw the git flow diagram and

Submit your .txt log file “mylog.txt” with you assignment.

**Hint: Script used in the video**: [**https://vimeo.com/138418055**](https://vimeo.com/138418055)

Git Merging Demo Steps

Create SomeClass

Add method fnOne [syso: I couldn't repair your brakes, so I made your horn louder]

Run

git bash

cd to directory

git init

git status

git add \*.java

git status

git commit -m "Initial"

chg fnOne to thoughtForTheDay

git status

git commit -a -m "Refactored fnOne"

git checkout -b addEvents

add upcomingEvents [syso: Party at Jane's house tomorrow]

git commit -a -m "Added events"

add header to upcomingEvents

try: git checkout master, see error

git commit -a -m "Refined events"

now: git checkout master

revise thought for the day: On the other hand, you have different fingers

git commit -a -m "New thought for the day"

git branch --no-merged (see addEvent)

git merge addEvents [ success! different parts of the file]

git branch --no-merged

git branch --merged

git checkout -b moreEvents

Modify program:

variable:

private ArrayList<String> events = new ArrayList<String>();

new method:

public void createEvents() {

events.add("We're going to a movie on Saturday");

events.add("Study session on Sunday - Jim's house");

}

change method:

public void upcomingEvents() {

System.out.println("Upcoming Events");

for (String event : events)

System.out.println(event);

}

call in main:

sc.createEvents();

git commit -a -m "Add multiple events"

git checkout master

modify upcomingEvents ["Dinner at Katie's on Friday"]

add SomeClass sc = new SomeClass() to main

git commit -a -m "Different event"

git merge moreEvents [ conflicts! need to resolve]

in Editor, notice the lines with issues, fix!

>> how? remove lines from head, remove lines with === and <<<

>> in general? first decide which to keep, make these kinds of

>> changes.

git branch --no-merged

git merge [ won't let you yet!]

git commit -a -m "Merged event handling"

git branch --no-merged

git branch --merged

git checkout moreEvents

modify: Upcoming Events - Please join us!

git commit -a -m "More friendly events"

git checkout master

press up-arrow, git branch --no-merged

git merge moreEvents [success! it's only a conflict if 2 changes]

git log

git log -p -2

git log --pretty=oneline

git log --pretty=format:"%s" --graph

git log --pretty=format:"%s" --graph > mylog.txt

git config --global alias.gr 'log --pretty=format:"%s" --graph'

tomorrow

**Solve following Git Reciepe Problems: Please give explanation and git commands to accomplish the tasks assigned**

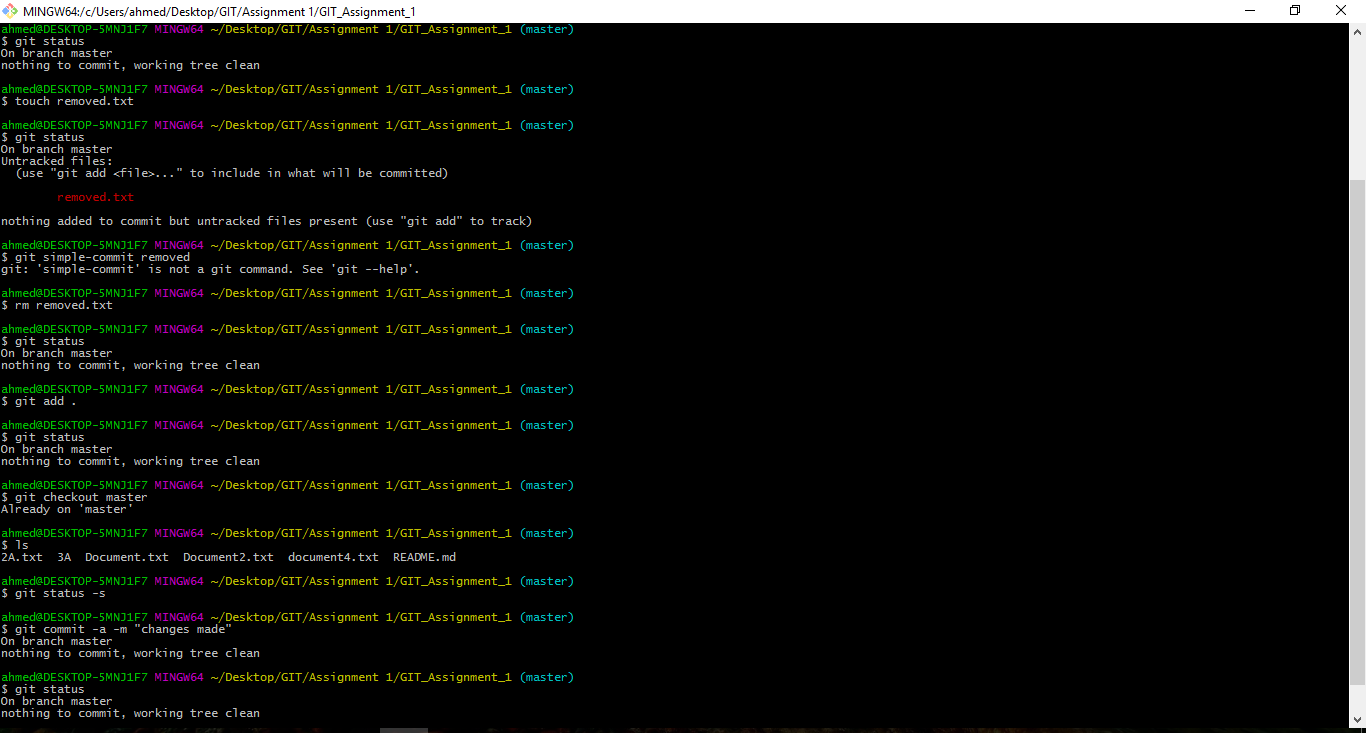
1. Committing a file removed with the standard rm command

Problem: Your repository is in a clean state and contains the removed.txt file. The file is committed. You want to remove the file using the standard $ rm command and then commit this modification into the repository.

**Sol:**

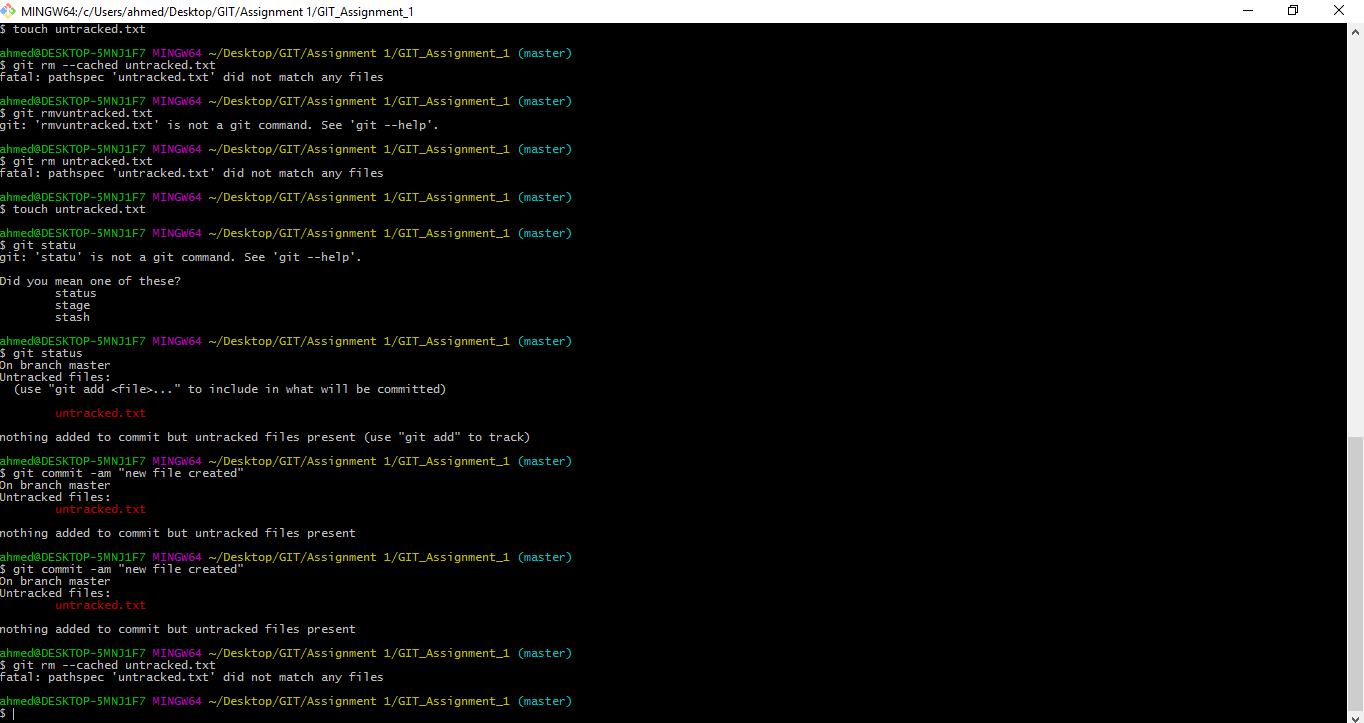
* creat a file removed.txt
* “rm removed.txt” command to remove the created file.
* check the status and the file is created and removed.
* checkout the file it gave error message saying there is so such file to checkout.
* check status of the file git status -s we get \_D removed.txt.
* Commit and check the status.
* There is no file removed.txt.

2. Converting an unmodified file into an untracked file  Problem:  The repository is in a clean state and the working directory contains one file—untracked.txt. The file is unmodified. You want to convert it into an untracked state.



3.Committing in a wrong branch

You have just created a new revision only to find out that it should go to a different branch. If your modifications do not collide while switching branches you can easily forget to check out the appropriate branch before you commit. After creating the revision in the wrong branch you want to move it to its correct destination.

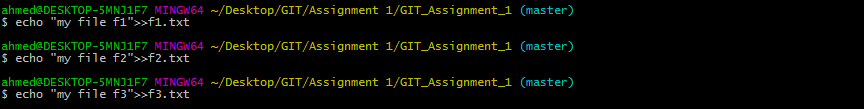


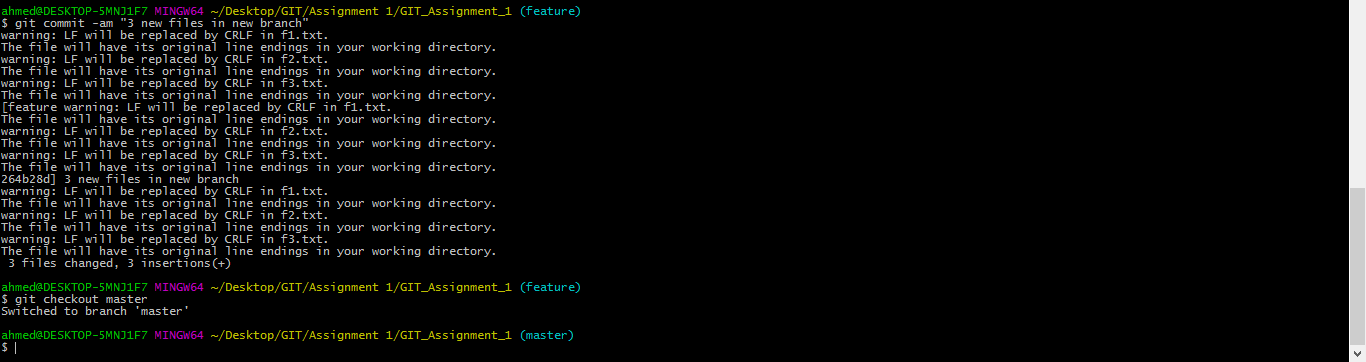
4. Merging diverged branches

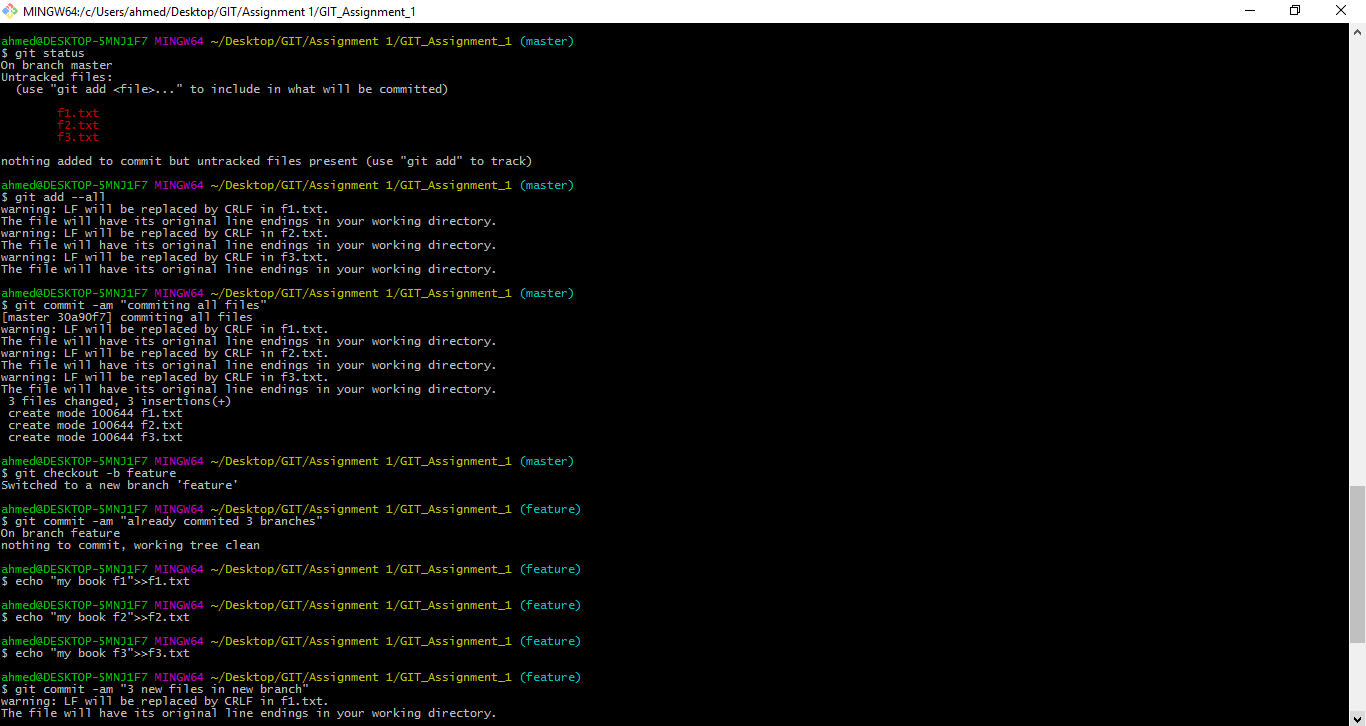
You want to merge the branches shown in Figure 6-5. The feature branch is to be merged into the master branch. The repository you want to obtain is presented in Figure 6-8. Figure 6-8 underlines the order in which revisions m4, m5 and f1, f2, f3 were created. In this recipe, this order is not important, therefore Figure 6-8 could also be drawn as in Figure 6-9.

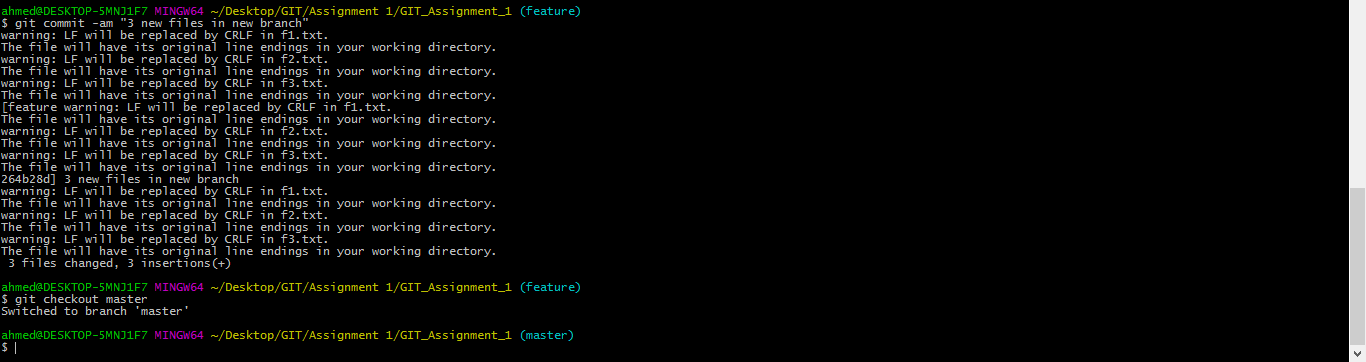
**Sol:**

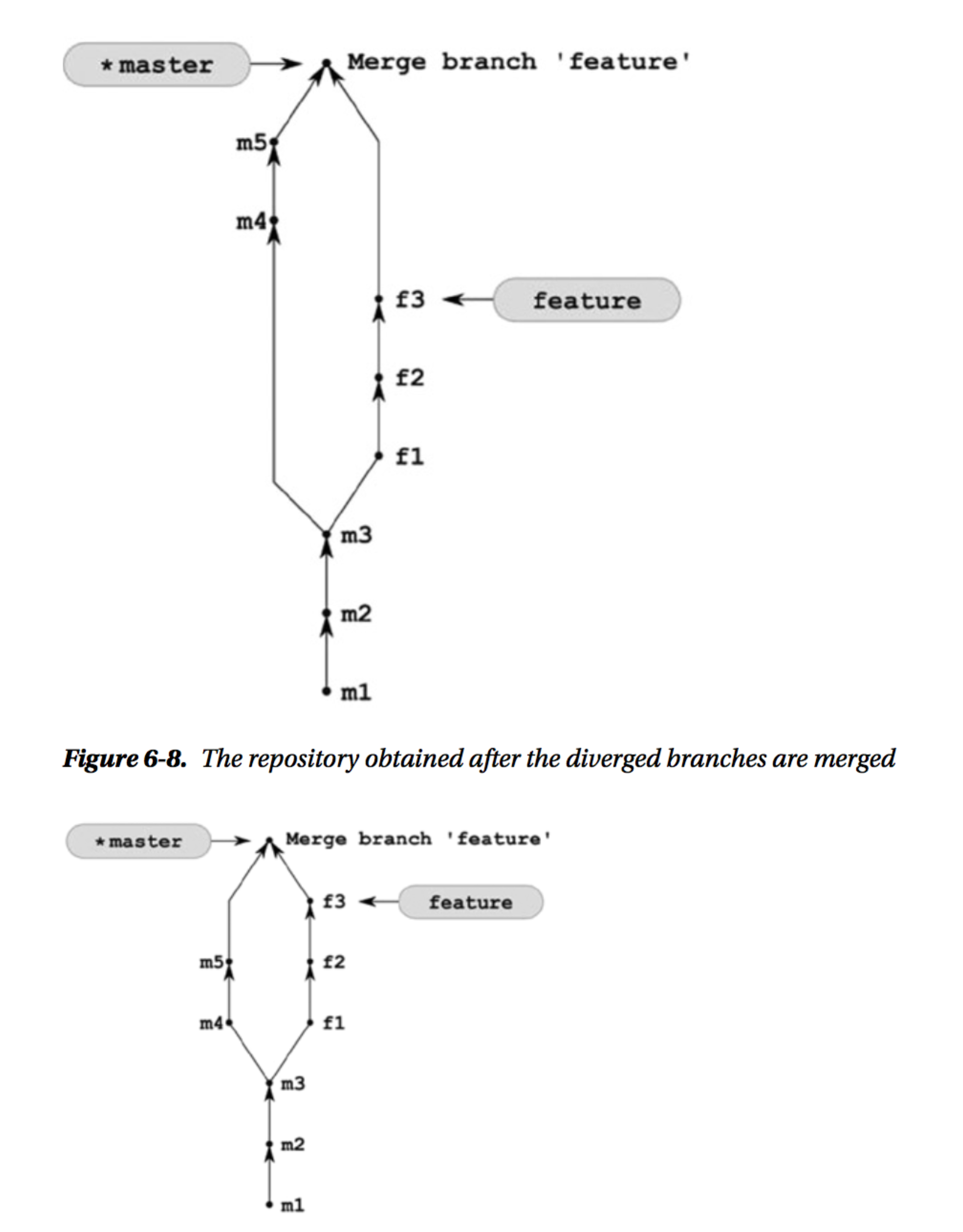
* Create a working directory
* Merge
* Create three files
* Committed them as m1, m2, m3
* Create a feature branch and it should have three commits.
* Use command git checkout -b feature
* Commit all three f1 f2 and f3 commits.
* Checkout master.



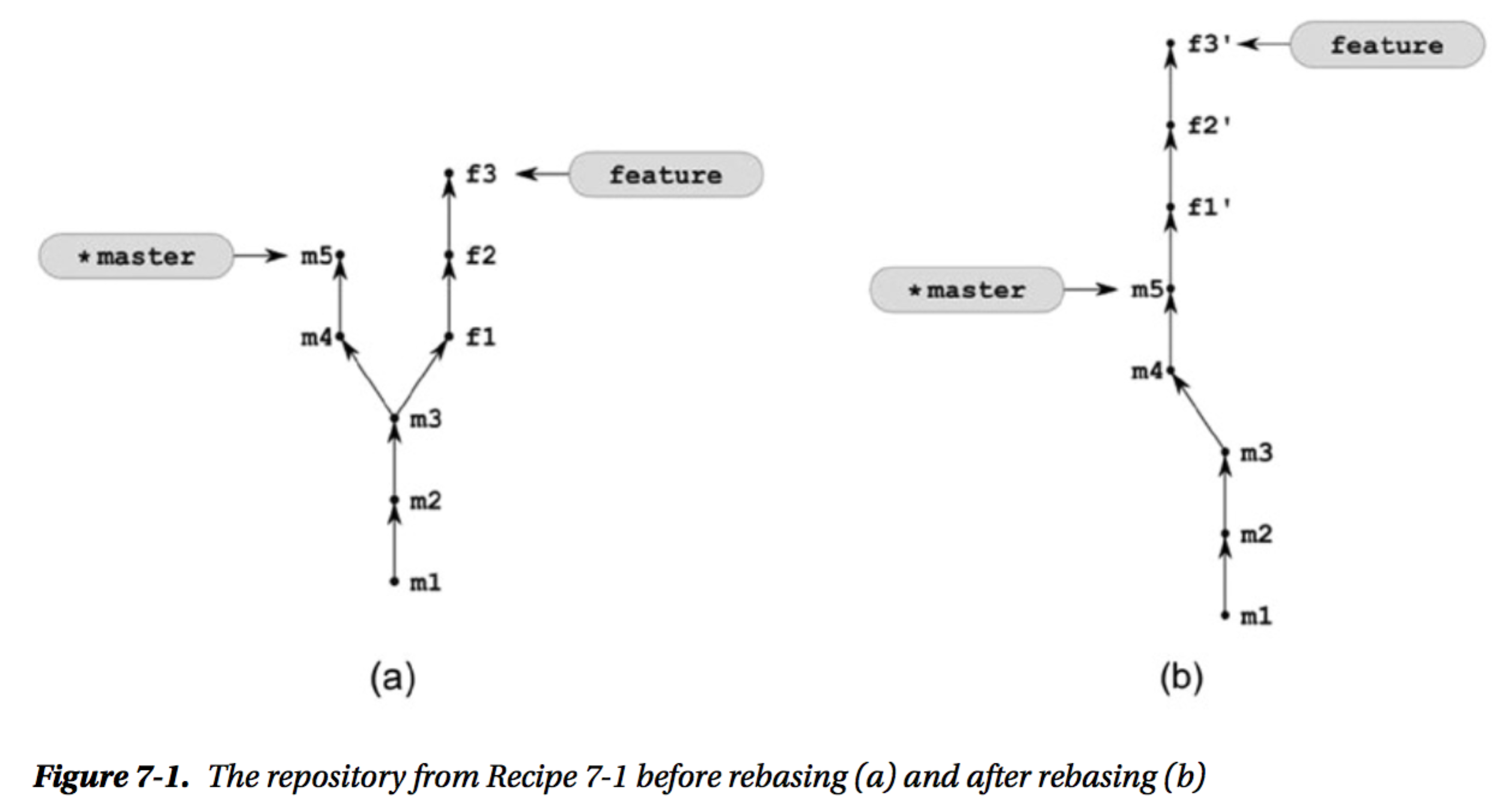








1. Rebasing divergent branches  Problem: You work in a repository with two branches named master and feature. The branches diverged and your repository  now looks like Figure 7-1(a). You want to transform the feature branch in such a way that:
   1. The history is linear (that means that the branches are not divergent anymore).
   2. The master branch is merged into the feature branch.
   3. All of the commits that were made in the feature branch are at the very top of the master branch.  The repository you want to achieve is presented in Figure 7-1(b).



**Sol:** Cloned previous repository mergdev working directory to repos1 directory. Then went to directory repos1 and then checkout the feature branch and then rebase the master.

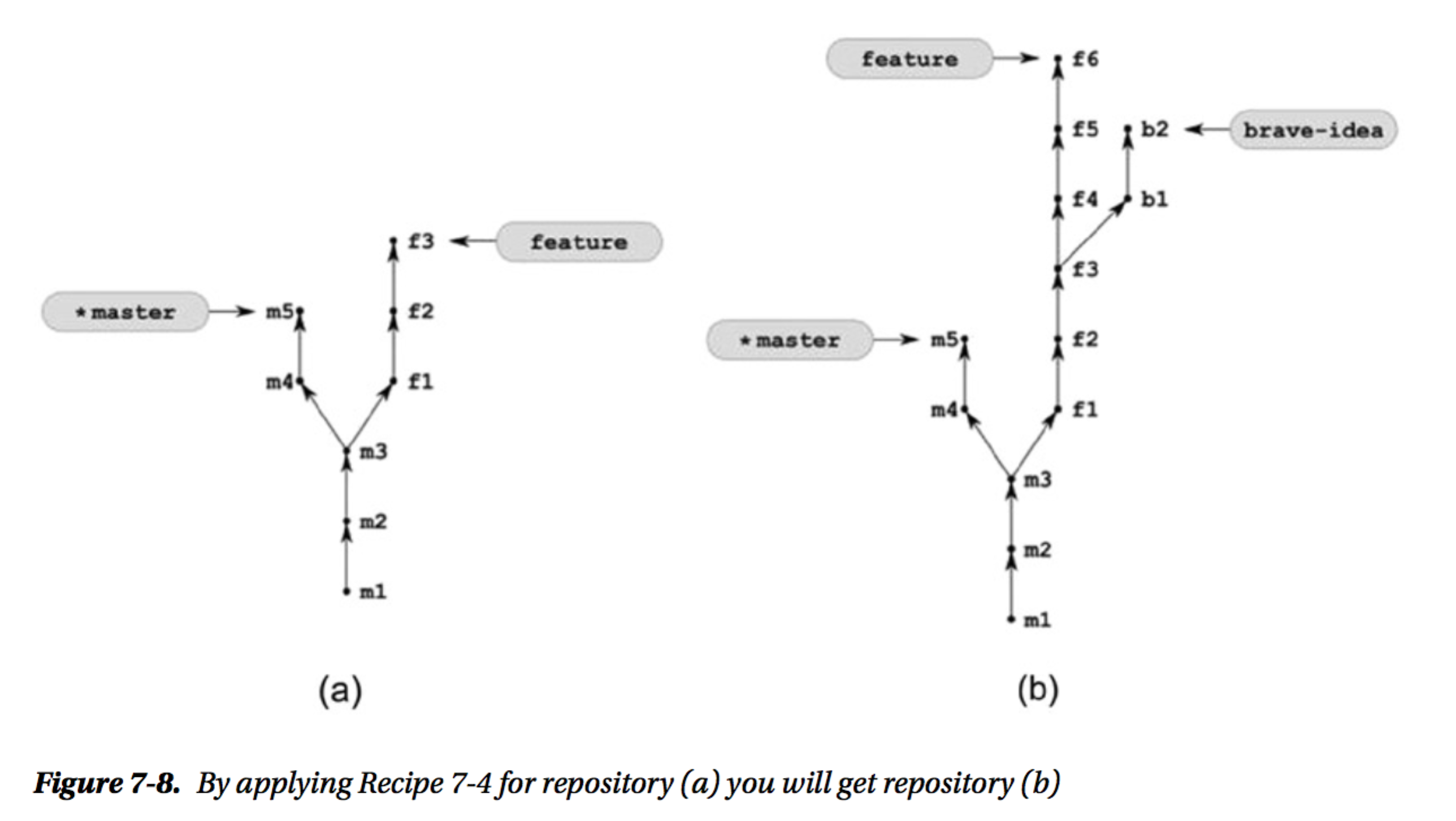
Command used to list all the reflog commits with comments containing f3

$ git log --walk-reflogs --grep=f3 --pretty="%h %s %cd" | sort | uniq

6. Diverging three branches Problem

Your repository contains two divergent branches master and feature, as shown in Figure 7-8(a). First you want to work on some new idea, basing your work on the latest revision in your feature branch. You need to create a new branch called brave-idea and to commit your changes as revisions b1 and b2. Next you want to switch to the feature branch and create three new revisions f4, f5, and f6. The repository you want to achieve is shown in Figure 7-8(b).

Figure on the next page…..



**Sol:**

Cloned previous work from previous answer merged and repos1 repositories and performed all changes given below

* Create brave-idea branch and checkout

$ git checkout -b brave-idea feature

* Create 2 revisions in brave-idea branch

$ git simple-commit b1 b2

* Checkout the feature branch

$ git checkout feature

* Create 3 revisions in the feature branch

$ git simple-commit f4 f5 f6

* Change the current branch to master

$ git checkout master