Git Lab 5 Branching and Merging

GT-WU-A-05 - Branching and Merging

By the end of this lab you should be able to:

- Create branches in Git
- Add files to branches
- Checkout branches
- Merge into branches

Step 1 - Navigate back to your repository

- Before you begin, make sure you are within your repository on the command line or in git bash
  - cd c:/git_repos/myrepo

Step 1 - Create a branch

- Using the *git branch* command, list all available branches in your repository "myrepo"
  - *git branch*
- Using what you have learned, create, add and commit a file called master.txt to your Git repository.
  - *echo text > master.txt*
  - *git add master.txt*
  - *git commit -m "commit message"*
- use *git branch* to create a new branch - name this *mybranch*
  - *git branch mybranch*
- Now rerun the *git branch* command to check it has been created
  - *git branch*
- Has the output changed?

Step 2 - Checkout the Branch and Make Changes

- use the *git checkout* command to switch to your newly created branch.
  - *git checkout mybranch*
- Using what you have learned, create, add and commit a file called branch.txt to your Git repository.
  - *echo text > branch.txt*
  - *git add branch.txt*
  - *git commit -m "commit message"*.
- Now switch back to your master branch
  - *git checkout master.*

Step 3 - Merge between branches

- The 'master' branch is the currently selected branch within your Git repository
- Check the files that exist in your workspace, you should see that "branch.txt" does not exist.
- Use the command *git merge t*o merge both branches together (and therefore all different changes between file(s)).
  - *git merge mybranch*
- Git should output when the merge completes, can you now see the file branch.txt?

Step 4 - Dealing with conflicts

- Create a new empty file called conflict.txt within your Git repository then add and commit it.
  - *echo text > conflict.txt*
  - *git add conflict.txt*
  - *git commit -m "commit message"*.
- Create a new branch called 'mybranch2' and switch your working directory to this branch.
  - *git branch conflictbranch*

    - *git checkout conflictbranch*
- Add the following line to the top of the file with your preferred text editor.
    - a=1
- Save the file, then add and commit your changes.

- Switch back to to the master branch
    - *git checkout master*
- Load the file you have just created in your favorite editor (it should NOT contain the change which you have just added as we are now on a different branch)
- at the top of the file add the content:
    - a=2
- Save the file, add and commit your changes.
  AtT this point the two branches have diverged into parallel development streams.

- Run the *git merge* command to carry out the merge operation with mybranch2
    - *git merge conflictbranch*
- When Git warns you of a conflicting change, open up the file to see how it is reported.

```
<<<<<<< HEAD
a=2
=======
a=1
>>>>>>> branch2
```

- This shows the conflict and which branch it came from. To fix this, leave this file with the contents which are purely 'a=2' and no other text.
- Save the file with these contents
- mark the conflict as resolved with git add
- *git add conflict.txt*
- Commit the change to complete the merge
- *git commit -m "merged from conflictbranch"*

Step 4 - Delete the branch

- Now that you have merged from *conflictbranch,* there is no more parallel development, we can safely delete the branch.
- First, make sure you are working on the master branch, if not :
    - *git checkout master*
- Then simply delete the reference
    - *git branch -d conflictbranch*
- If Git complains "Cannot delete the branch '*conflictbranch*' which you are currently on" then you have not switched to the master branch yet.
- If Git complains "*conflictbranch* contains work not included in the current HEAD" this means you have not completed the merge yet.

Optional Steps - Going Further

You are now going to simulate working on a feature in git, and squashing it into a single commit.

Step 1 - Create a feature branch

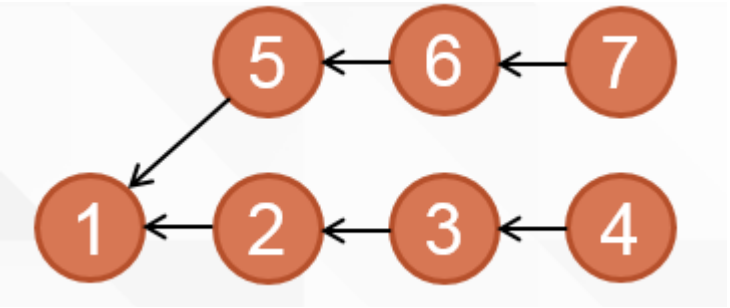
- Use what you have learned to complete the following steps:
    - Create a branch called *feature* (based on the master branch)
    - Switch the the *feature* branch
    - Create three new commits, each time creating one new file

Step 2 - Work on the master branch

- Switch back to the master branch
- Create three new commits, each time creating at least 1 new file

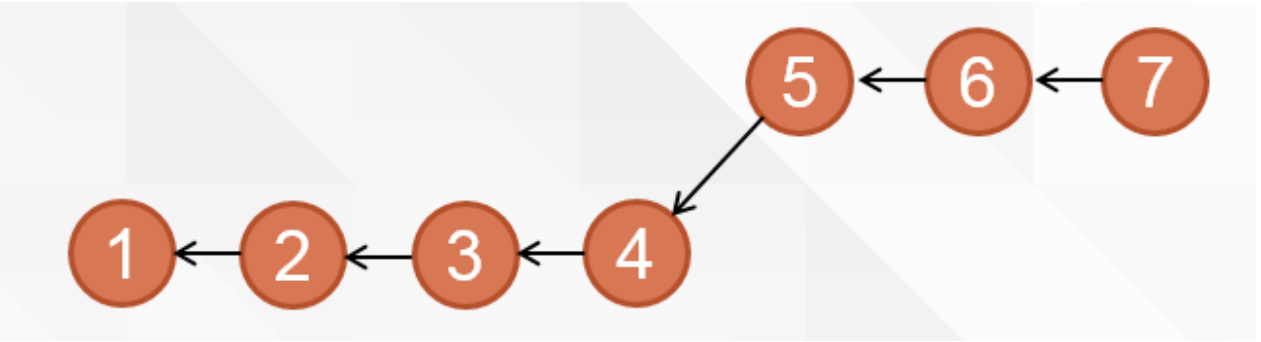
Step 3 - Rebase the feature branch

- In a real environment, those commits on the master branch may have been created by other people. as things are, other peoples features exist on the master branch which have not been included in your feature branch.
- We could merge those changes into our feature branch, but this would prevent us from squashing (you cannot squash a merge commit).
- Instead, we can rebase the branch, currently, our branches looks like this :



- *git checkout feature*
- *git rebase master*

- Our branches now look like this:



Step 4 - Squash the feature branch

- Next, before we merge our own changes we want to squash them into a single commit
  - *git rebase -i HEAD~3*
  - This command rebases our feature branch based on the commit three before the HEAD (in this case - commit 4)
- Your default text editor will pop up.
- Note that your own hash id's will be different, and your view will differ from the image below depending on your editor

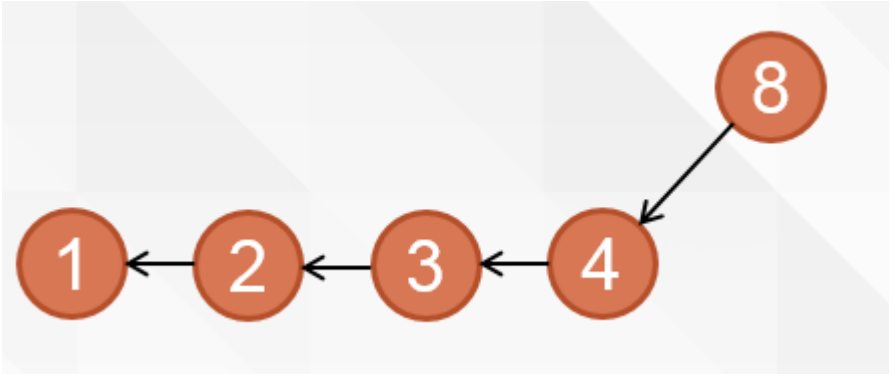
```
pick 7837635 finished feature x
pick 1ef8d0f added class in foo.c
pick 73c58e5 Created README.txt
pick b1b7e12 moved user guide to docs

# Rebase c27e33e..b1b7e12 onto c27e33e
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

- Change all these lines except the top one from *"pick"* to *"squash" (*or *"s"*)

```
pick 7837635 finished feature x
squash 1ef8d0f added class in foo.c
squash 73c58e5 Created README.txt
squash b1b7e12 moved user guide to docs
```

- Save your text editor and close it
- The rebase should progress automatically.
- Our branches now look like this:



- We can now merge into the master branch, which will become a fast forward merge
  - *git checkout master*
  - *git merge feature*
- And finally, delete our feature branch
  - *git branch -d feature*

When you have completed the steps above, it is time to move onto Module 6.