**Lab 6**

***Rebasing***

In the last lab we consider branching and merging with conflicts. In Git there are two ways to integrate changes from one branch into another: *merge* and *rebase*.

Rebase is similar to merging, except the other direction. Instead of merging a *branch* into *master* and possibly having conflicts, a rebase essentially merges *master* into a *branch*. When conflicts occur, they are moved to a temporary location. There, you can fix the conflicts (or abort), then continue the rebase. Finally, a merge is done to pull *branch* into *master* which should occur (fast-forward) with no conflicts.

The scenario will be the same as the last tutorial: (a) create a new branch, *fix1* off of *master*, (b) edit a file there, stage and commit to the branch, (c) create a branch, *fix2* off of *master,* (d) edit the same file there which conflicts with the edit in *fix1,* stage and commit the change, (e) do a rebase (there will be no conflicts), (f) merge *fix2* to in *master,* (g) go back to the *fix1* branch, rebase, which generates a conflict just as in the previous tutorial, (h) resolve the conflict, (i) continue the rebase, (j) and then merge *fix1* into *master*.

From my understanding, rebasing seems a better approach at resolving conflicts because it doesn’t affect *master*. Then, when resolved, you can merge the non-conflicting *branch* into *master*.

(Optional) A detailed explanation of exactly what is happening with a rebase is found here:

<https://git-scm.com/book/en/v2/Git-Branching-Rebasing>

For this lab, we will start from scratch.

**Steps to Complete – Part 1**

This shows an example of a [Competing line change merge conflicts](https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/).

1. Do the following:
2. Close Cmder or navigate out of the *gitex* directory.
3. Use File Explorer to delete everything in your *gitex* directory. It should be completely empty
4. Open Cmder and navigate to *gitex.*
5. **Here, we load up the *master* branch with working code**. Do the following:

**λ**  git init

**λ**  notepad foo.txt // Type the text: “Friday”

**λ**  git add foo.txt

**λ**  git commit –m “New foo in master”

1. **Next, Developer 1 is notified to fix a bug in the code. She creates a branch and does the fix.** Do the following:

**λ**  git checkout –b fix1

**λ**  notepad foo.txt // Change text to “Saturday”

**λ**  git commit -a -m "fixed foo in fix1"

1. **Next, Developer 2 is notified to fix a bug in the code. He creates a branch and does the fix.**. Do the following:

**λ**  git checkout master

**λ**  git checkout –b fix2

**λ**  notepad foo.txt // Change text to “Sunday”

**λ**  git commit -a -m "fixed foo in fix2"

1. **Developer 2 is finished and is ready to merge his code back into *master.* Assume Developer 1 is still working on her fix.** We use the rebase approach here, and it will succeed with no problem because the *master* branch has not been updated with anything conflicting (it hasn’t been updated at all, since the *fix2* branch was created). Do the following:

**λ**  git rebase master // Should say, “Current branch fix2 is up to date”

**λ**  git checkout master

**λ**  git merge fix2 // Should say, “Updating…Fast-forward...”

**λ**  git branch –d fix2 // All done with this branch now

**λ**  type foo.txt // Displays: “Sunday”

1. **Developer 1 is now finished and ready to merge her code back into *master*.** We use the rebase approach which will fail because *master* now conflicts with *fix1* because *master* has been update since *fix1* was created, and there is a conflict in the first (only) line of the file. Do the following:
2. Checkout *fix1:*

**λ**  git checkout fix1

1. Rebase with master. The result is shown below (message will be different, things have changed! But spirit is the same).

**λ**  git rebase master



This tells us several things:

1. There is a conflict in *foo.txt*. Thus, we need to edit this file and fix the marked-up conflict.
2. Notice the heading above the command prompt has changed, as shown below. This means that we are on a temporary branch.



1. We have several choices:
	* 1. “git rebase --continue” – Continues the rebase. If successful, the temporary branch will be removed and *fix1* will now have incorporated any non-conflicting changes in *master* and the current branch will be *fix1*.
		2. “git rebase --skip” – I’m not sure what this does.
		3. “git rebase --abort” – Aborts the rebase and the current branch will be *fix1*.
2. **Developer 1 fixes the conflict.** Open *foo.txt* and change it so that it says: “Saturday Sunday”

**λ**  notepad foo.txt

1. Add the change:

**λ**  git commit -a -m “fixed conflict in detached head”

1. Continue the rebase:

**λ**  git rebase --continue

Notice the heading above the prompt has changed back to *fix1* since the rebase is complete (your display will be different, *e.g.* the “Applying…” message will be something different):



1. **Developer 1 merges *fix1* into *master*.** Do the following:

**λ**  git checkout master

**λ**  git merge fix1

**λ**  git branch –d fix1

**λ**  type foo.txt // Displays: “Saturday Sunday”

1. **Do the following:**
2. Make a screen shot of the top 3 commits of: git log (in *master*). Make sure it shows your name and date.
3. Place the image in the *HW VCS* document in the appropriate place.
4. The image should easily readable without zooming in or out.

Good idea: make a backup copy of your *gitex* folder with the name: *gitex\_6*. Then, you’ll start Lab 7 using the *gitex* repo. Lab 7 does continue from Lab 6.