**Lab 4**

***Branching and Merging***

So far you have worked off the *master* branch of your repository. Git allows you to make a copy of the master branch, work off that copy, and then merge your changes back into the *master* branch. Or, in general, you can create a branch off of any branch.

When you work with Git, best practice is that the code in your *master* branch should always be working and tested code. In other words, you don’t directly edit the code there. Instead, you create a branch off of master and work there. When the code in the branch is finished and tested, then you merge the branch into *master*. In other words, when you want to start work on a new feature you don’t want to risk destroying the working code in the *master* branch, so you do the new development on a branch.

When you work in a group everyone should have their own branch, periodically merging back into *master*. If a group member merges into *master* you can use Git to pullthose changes to your branch, if needed. We illustrate this in Lab 8

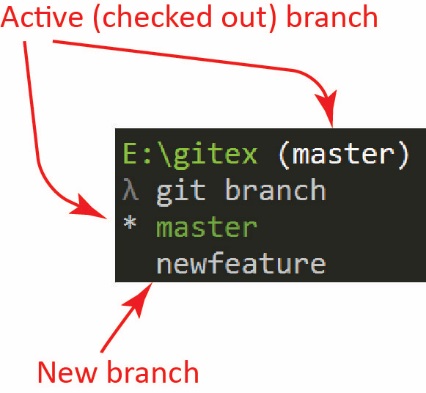
In this lab you will create a new branch, work on files in the new branch and then merge the changes back to the *master* branch. This lab continues from Tutorial 3.

**Steps to Complete**

1. Do the following:
2. Create a new branch named *newfeature*:

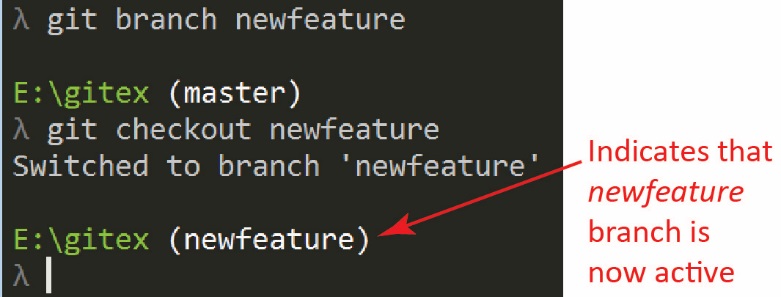
**λ**  git branch newfeature

This command copies the files in the current branch (which in this case is *master*) to the new branch. You might be more familiar with a syntax like: *copy from to*; however, Git does it as shown above.

1. Display a list of all the branches:

**λ**  git branch

The result is shown on the right.

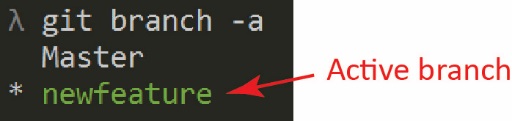
1. Switch (checkout) to the new branch:

**λ**  git checkout newfeature

The result is shown on the right.

1. (Read, no action required) You can create a branch and check it out with a single command with this syntax:

**λ**  git checkout -b myNewBranch

1. Display a list of all branches:

**λ**  git branch

The result is shown on the right.

|  |
| --- |
| **Warning**: Somehow “master” got changed to “Master” in the figure above. This probably resulted from developing these tutorials over and over. Almost certainly yours will say “master”. |

1. Create a new file, *bar.txt* and add the text, “This is a new file”

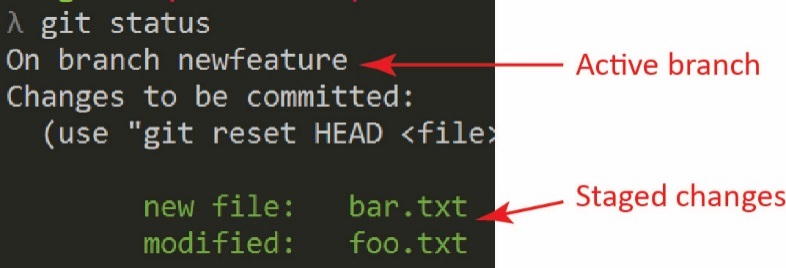
**λ**  notepad bar.txt

1. Open *foo.txt* and add this text to the end of the file: “Welcome back.”, then save.

**λ**  notepad foo.txt

1. Stage both files (can also use: git add –A, not sure the difference but is the same for us)

**λ**  git add .



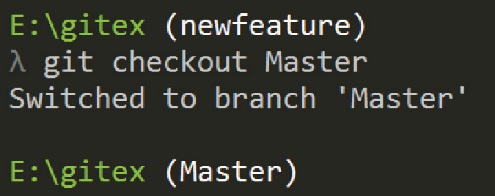
1. Check the status:

**λ**  git status

The result is shown on the right.

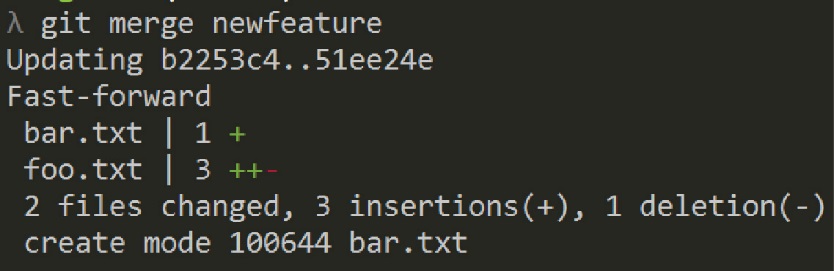
1. Commit the changes to the *newfeature* branch:

**λ**  git commit -m "Expanding helloworld app"

1. Next, we merge the *newfeature* branch with the *master* branch. The way this works is you have to switch to the branch that you want to merge into (*master*) and then issue the *merge* command specifying the branch you want to merge (*newfeature*).Follow these steps:
2. Checkout the master branch

**λ**  git checkout master

Note: the figure shows “Master” while yours is probably “master.”

1. Merge the *newfeature* branch

**λ**  git merge newfeature

1. Display the folder contents and verify that the new file, *bar.txt* is present. Remember, you are on the *master* branch now so we hope to see that *bar.txt* has been added:

**λ**  dir

1. Display the contents of *foo.txt* and verify that the changes are there:

**λ**  type foo.txt

1. Display the commit log and note that the commit from the *newfeature* branch is now apart of the commits for the *master* branch as shown on the right

**λ**  git log –n 3

1. (Read, no action required) This was an “easy” merge as indicated by “Fast-forward” in the figure from Step 7b above. What made it easy was that *master* had not been committed to after the *newfeature* branch was created. In the figure on the left below, before the merge we have committed to *newfeature* (C2) and it is directly ahead of *master.* After the merge, Git simply moves the *master* pointer forward as shown on the right below.

|  |  |  |
| --- | --- | --- |
| Before merge |  | After merge |
| E:\Data-Grant\IndirectFundProposal_2017\Git Stuff\ver2\b2.jpg |  | E:\Data-Grant\IndirectFundProposal_2017\Git Stuff\ver2\b3.jpg |

A more complex situation occurs when there are other branches that have been created and merged into master. Suppose the *newfeature* branch is created. Next, suppose the *hotfix* branch is created, a file, say *f1* is modified and committed (see figure on the left below). Next, suppose *hotfix* merges into *master* (see figure on the right below)*.* This is a fast-forward merge. Finally, suppose *newfeature* is ready to merge into *master*; however, *newfeature* is no longer directly ahead of *master*. In other words *f1* in *newfeatue* may be in conflict with *f1* in the new *master.* Note that the figure on the right below does not show *newfeature* being merged into *master, i*n a later tutorial we will consider merge conflicts.

|  |  |  |
| --- | --- | --- |
| Before merge |  | After *hotfix* merge |
| E:\Data-Grant\IndirectFundProposal_2017\Git Stuff\ver2\b4.jpg |  | E:\Data-Grant\IndirectFundProposal_2017\Git Stuff\ver2\b5.jpg |

1. Consider this scenario: suppose a file is modified on *master* and now you want to “overwrite” it in *newfeature,* so that you have the most recent version*.* One way to do this (from the *newfeature* branch) is: git checkout master filename. The resulting file will be added to the index (staged) in *newfeature.* Then, you would commit it. Follow these steps:
2. Checkout the master branch (you probably already have it checked out)

**λ**  git checkout master

1. Open *foo.txt* and add this test at the end: “Roundabout”

**λ**  notepad foo.txt

1. Stage and commit the file

**λ**  git commit –a –m “New feature to app”

1. Checkout *newfeature*

**λ**  git checkout newfeature

1. Copy foo.txt into *newfeature*:

**λ**  git checkout master foo.txt

1. Commit the file:

**λ**  git commit –m “updated from master”

1. Verify the contents of *foo.txt* in *newfeature:*

**λ**  type foo.txt

1. (Read, no action required) This technique will work for (a) new files added to *master* and (b) more complex changes to *foo.txt* in *master* (as far as I can tell!). Thus, it looks like: git checkout branch file, essentially replaces any existing file in the active branch, but still must be committed.
2. Delete the *newfeature* branch:
3. Checkout the master branch

**λ**  git checkout master

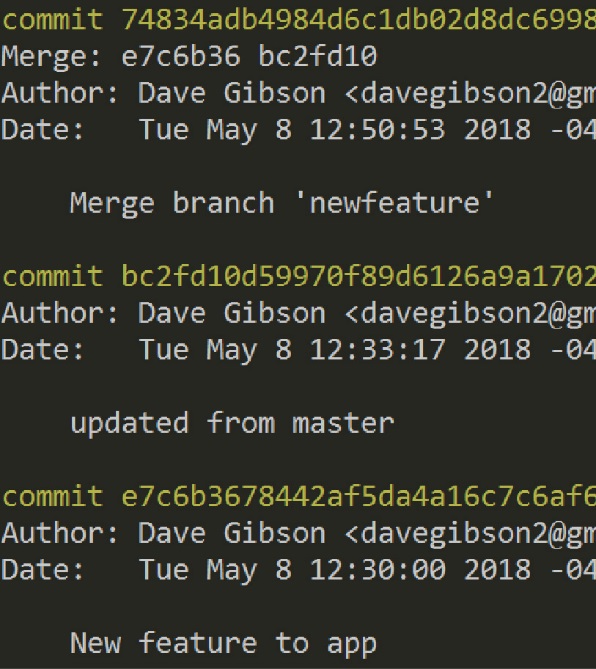
1. (Read, no action required) Generally, we would just delete the branch. However, since we copied *foo.txt* from *master* in the Step 9 and committed it to *newfeature, newfeature* should be merged first.
2. Merge *newfeature:*

**λ**  git merge newfeature

1. Delete the *newfeature* branch

**λ**  git branch –d newfeature

1. (Read, no action required) We could have forced the deletion, without merging by using: git branch –D newfeature



1. Verify that *newfeature* is deleted

**λ**  git branch

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\_4*. Then, you’ll start Lab 5 using the *gitex* repo.