

COMP390: Git Tutorial

Adrian Manhey

February 2022

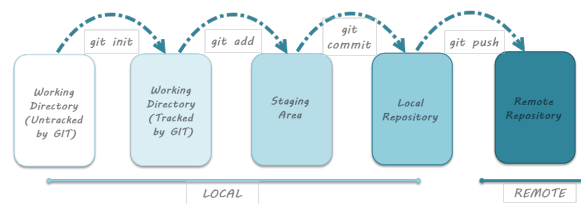
1 Introduction

The purpose of this tutorial is to allow someone who has installed Git to understand how to use it. If you have not yet installed Git, you can follow the tutorial here in order to do so.

2 Git Overview

First let's quickly go over the four data stores of Git. There are four parts in the Git life-cycle:

- Working tree – files currently on your computer
- Staging area – files/changes ready to be committed
- Local repository – files (and their history) that have been committed
- Remote repository – files (and their history) on a different computer



We will go over the commands that move documents from each stage later on this tutorial but have these components in mind as you follow along.

3 Getting Started

To begin we need a Git Repository, which can either be created on your local machine or cloned from an existing repository. If you wanted to clone an existing repository you would navigate to the directory you wanted the repository in via the command line and then use the command:

```
$ git clone <repository URL>
```

However, in this tutorial we are going to create our own repository (repo). To begin navigate to the directory you want to create your repo in. Once there, let's create a new README.md file for our repo. One way to do this is using the "echo" command:

```
echo "# COMP390Tutorial" >> README.md
```

This command saves the text in quotations in the README.md file on the last empty line. If we wanted to overwrite the file we would use a single ";" instead of two. This newly created file exists on the working tree. Now let's initialize the repository:

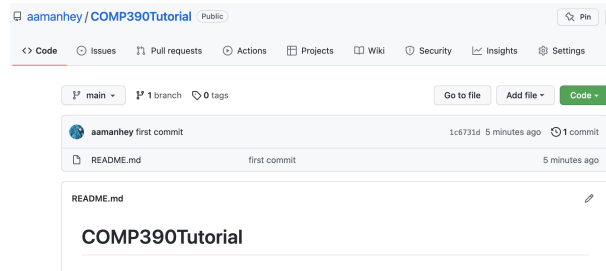
```
git init
```

Now we have to begin tracking our files to save the changes that have been made. The "git add" command adds a file to the list of files we want to begin tracking, files need to be added anytime a set of changes have made to them that we want to save. This moves our file from the working tree to the staging area. Then the "git commit" command saves the changes locally and moves them from the staging area to the local repository.

```
git add README.md  
git commit -m "first commit"
```

Next we begin moving our file from the local repository to the remote repository. The "git branch" command tells git which branch we want to work on, since this is the only file so far it's going to be on the "main" branch. Then we link our local repo to the remote repo with the next command, so git knows where to save our file. Finally we push our files to the remote repo.

```
git branch -M main  
git remote add origin https://github.com/<GitHub username>/  
    <repository name>.git  
git push -u origin main
```



Now that our files are in the remote repo we can view them on GitHub by navigating to your repo. You should see a "README.md" file with "COMP390 Tutorial" inside.

4 Changing Files

Let's make some changes! First let's add onto our current README file and then create an outline.txt file to document our project.

```
echo "Additional info about our project." >> README.md
echo "Project Outline" > outline.txt
```

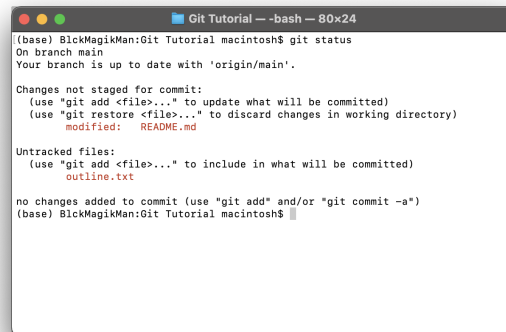
If we wanted to move both of them to the staging area we could use the "A" tag with the add command but let's just add the README file.

```
git add README.md
```

To check to see which files have been changed we can use the following command

```
git status
```

This should return something similar to this:



```
Git Tutorial --bash-- 80x24
(base) B1ckM@g1kM@n:Git Tutorial macintosh$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

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

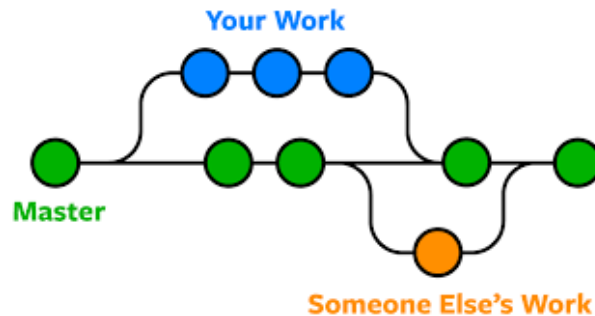
no changes added to commit (use "git add" and/or "git commit -a")
(base) B1ckM@g1kM@n:Git Tutorial macintosh$
```

Now let's commit and push our file.

```
git commit -m "adding to README"
git push
```

5 Stashing and Branching

Now let's say that we wanted to add a new feature to our project. It is better to reserve finished items on the main branch so that if more than one person is working on it they can branch off without having to wait for you to finish. You can think of branching as working on your own copy of the code that will later be added to the main line of code.



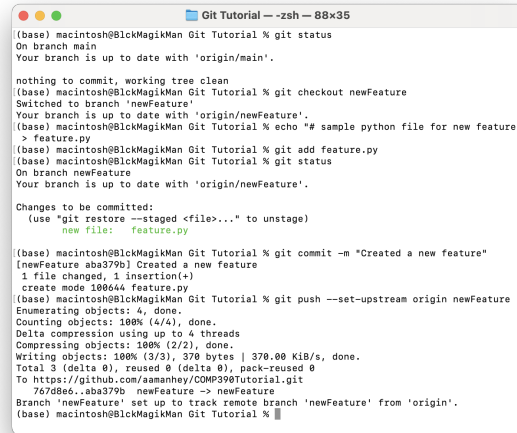
In order to branch we have to create one and then switch to it. Currently, we should have untracked changes to `outline.txt` so in order to switch branches without deleting those changes, we need to stash them. First, the changes need to be tracked in order to stash them then we can use the "git stash" command to stash the changes to `outline.txt`. Now to check that our file has been stashed we can use "git stash list". Now you should have a clean working tree, which can be checked with "git status". To move to another branch we have to create the branch and then check it out:

```
git branch newFeature
git checkout newFeature
```

or use the shortcut

```
git checkout -b newFeature
```

Here lets create a python file that is going to represent our new feature.



```
(base) macintosh@81ckMagikMan Git Tutorial % git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(base) macintosh@81ckMagikMan Git Tutorial % git checkout newFeature
Switched to branch 'newFeature'
Your branch is up to date with 'origin/newFeature'.
(base) macintosh@81ckMagikMan Git Tutorial % echo "# sample python file for new feature" > feature.py
(base) macintosh@81ckMagikMan Git Tutorial % git add feature.py
(base) macintosh@81ckMagikMan Git Tutorial % git status
On branch newFeature
Your branch is up to date with 'origin/newFeature'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   feature.py

(base) macintosh@81ckMagikMan Git Tutorial % git commit -m "Created a new feature"
[newFeature aba379b] Created a new feature
1 file changed, 1 insertion(+)
create mode 100644 feature.py
(base) macintosh@81ckMagikMan Git Tutorial % git push --set-upstream origin newFeature
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 370 bytes | 370.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/aamanhey/COMP390Tutorial.git
767d86c..aba379b  newFeature -> newFeature
Branch 'newFeature' set up to track remote branch 'newFeature' from 'origin'.
(base) macintosh@81ckMagikMan Git Tutorial %
```

Now that we've created the new feature let's go back to the main branch and merge the newFeature branch. This lets you take the independent paths of development created by the branches and combine them into a single branch, effectively bringing the changes we made on the newFeature branch onto the main branch.

```
git checkout main
git merge newFeature
```

If we were to continue to develop the new feature we could keep the newFeature branch but in this case, let's delete it:

```
git branch -d newFeature
```

Now to bring back our stashed changes we can use the **git stash apply** command to bring back the changes that were last stashed. It would be good practice to unstash our outline and then push those changes but it is not necessary for this tutorial. Since we've pushed a change from a branch other than main, newFeature, we need to create a pull request on GitHub to finalize the incorporation of our changes.

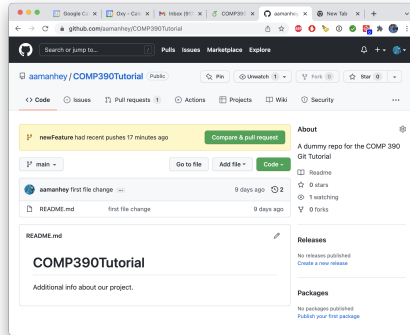


Figure 1: Push Notification

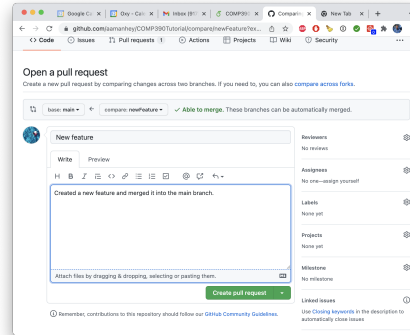


Figure 2: Open Pull Request

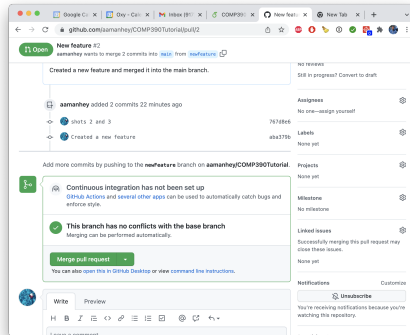


Figure 3: Merge Pull Request

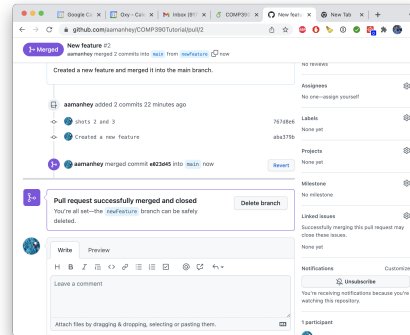


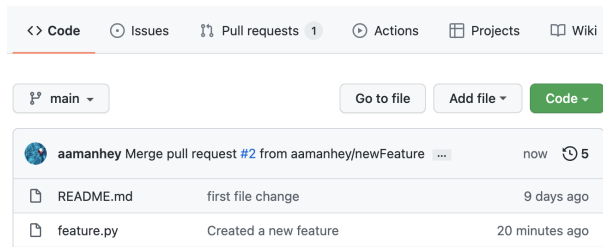
Figure 4: Successful Merge

6 Pull Requests

This part of the tutorial will show you how to make pull requests on GitHub. Open up your repository on GitHub. You should see a notification that recent pushes have been made to the main branch (Figure 1).

If you click on the **Compare & pull request** button it should navigate you to an open pull request page where you can add a description of your pull request (Figure 2). Once you create the pull request you'll be shown all the commits that have been made to that branch before and GitHub will check to see if your code can be merged into the main branch or if a conflict must be handled. You shouldn't see any merge conflicts so you should be able to press **Merge pull request** (Figure 3). The page should change and you should now see a message that your pull request was successfully merged (Figure 4).

Now when you go back to your repository's code you should see your feature.py file.



Congratulations! You have successfully created a GitHub repository, created and modified file, stashed and unstashed changes, created, used, merged, and deleted branches, and created a pull request!