# **ASSIGNMENT #1**



# **Submitted by Ahsan Javed (#153196)**

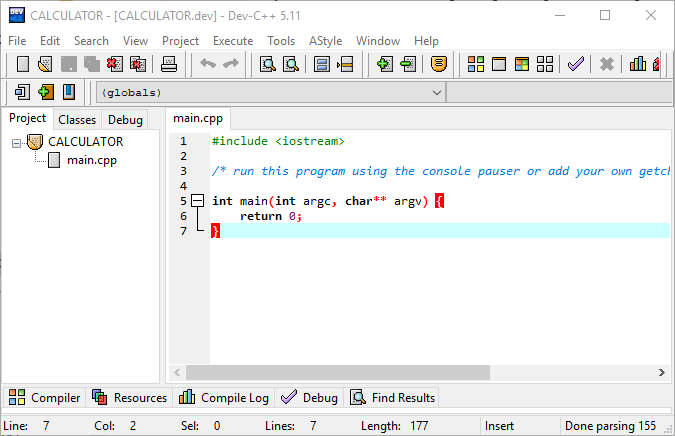
# **GitHub: @iAhsanJaved**

# **Submitted to Sir Ahmad Mohsin**

# **Git and GitHub Exercises**

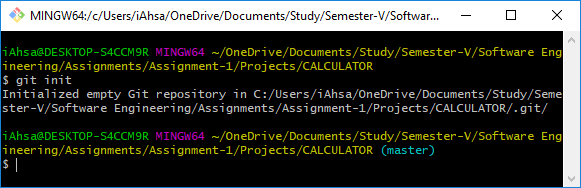
# EXERCISE-0: Putting A C++ Project on GitHub

**Step 1**: Create an A C++ PROJECT IN DEV OR CODEBLOCKS. I called mine CALCULATOR.

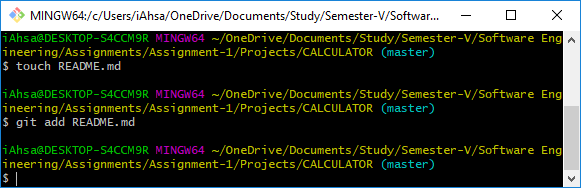


**Step 2**: Create a repository on GitHub of the same name.

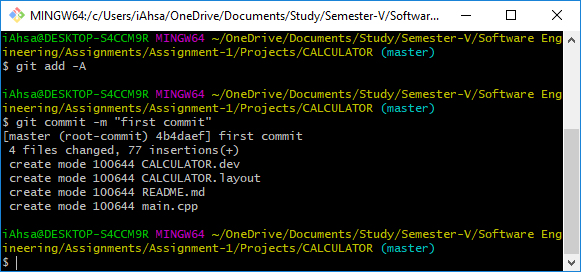
**$ git init**



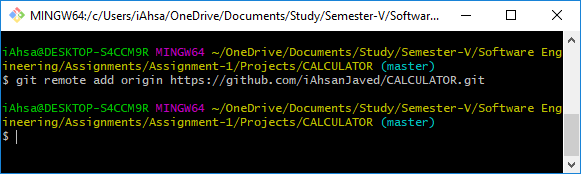
**$ git add README.md**



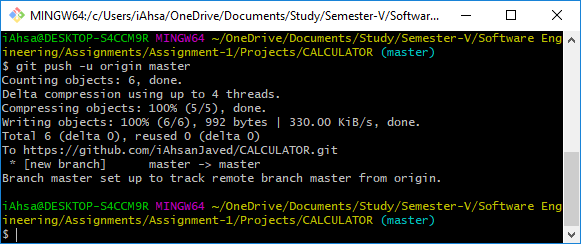
**$ git commit -m "first commit"**



**$ git remote add origin https://github.com/iAhsanJaved/CALCULATOR.git**

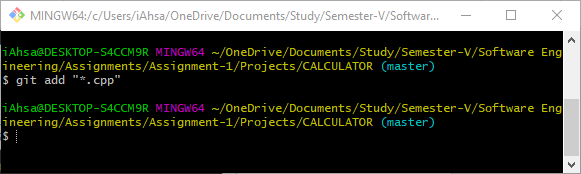


**$ git push -u origin master**

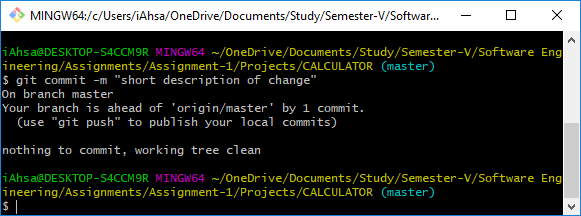


After a change to your project:

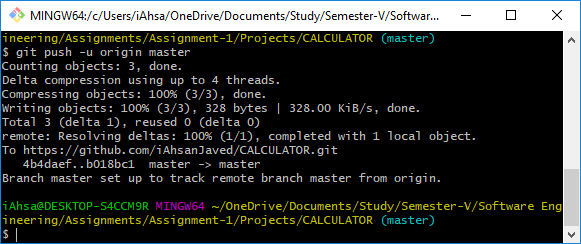
**$ git add –A**



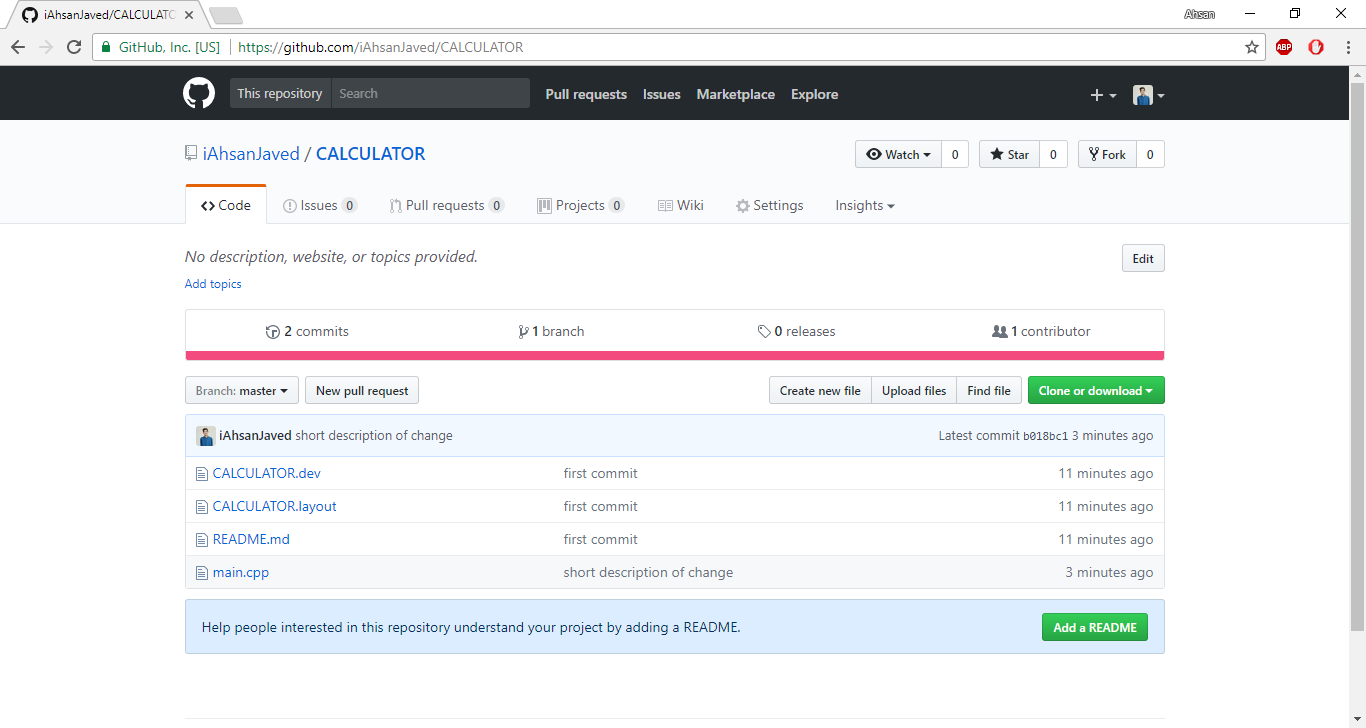
**$ git commit -m "short description of change"**



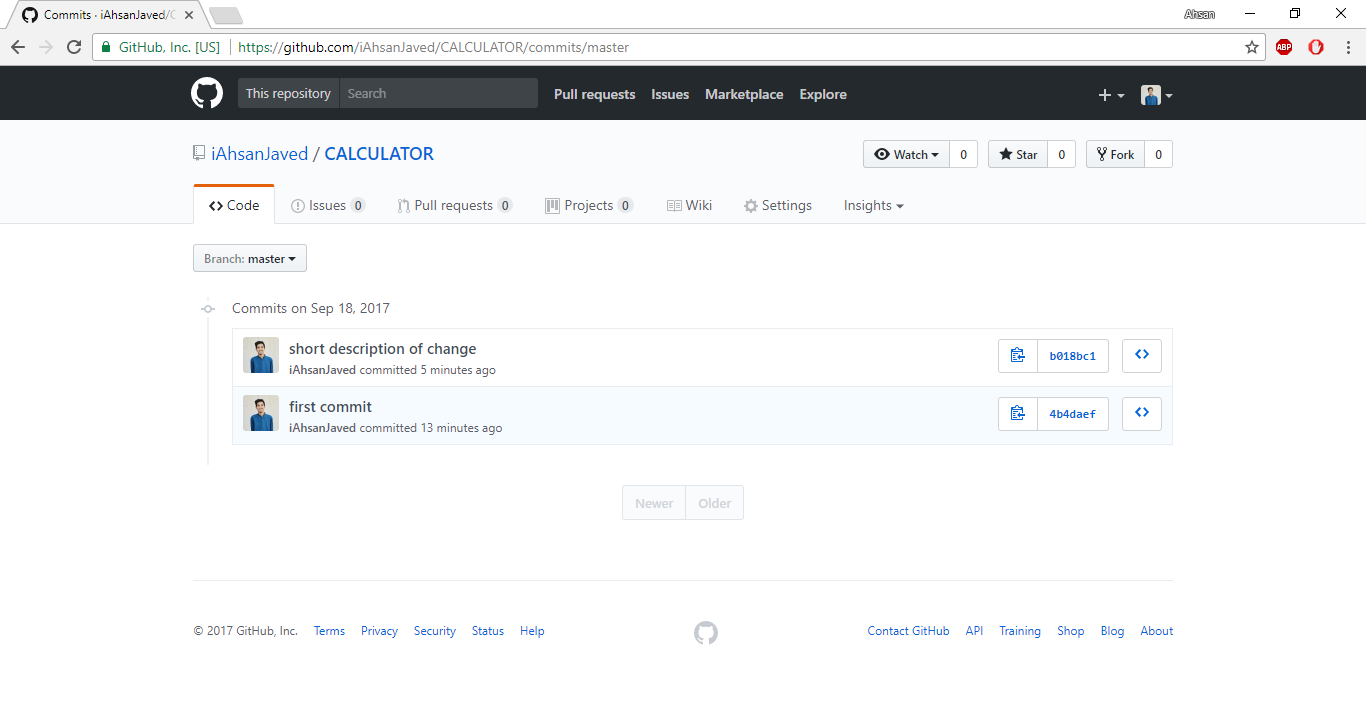
**$ git push -u origin master**



**CALCULATOR Repository on GitHub**  
URL: https://github.com/iAhsanJaved/CALCULATOR



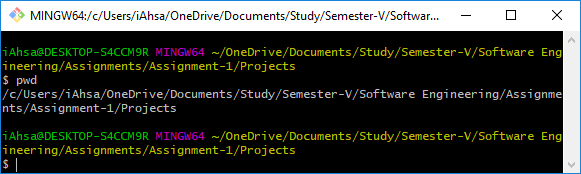
**Commits on CALCULATOR Repository**



# EXERCISE-1: Creating and using a local repository

Show your current directory (pwd = print working directory):

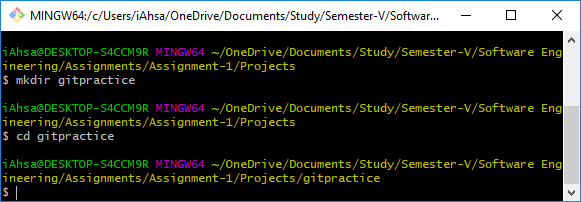
**$ pwd**



Create a directory:

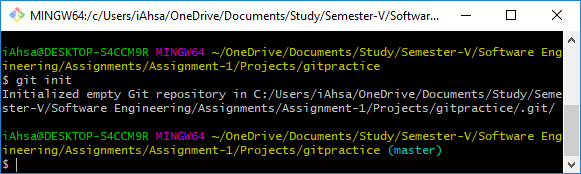
**$ mkdir gitpractice**

**$ cd gitpractice**



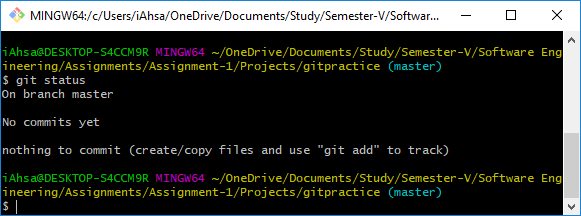
Start tracking files and subfolders with git:

**$ git init**



Run git status to see current status of the repository:

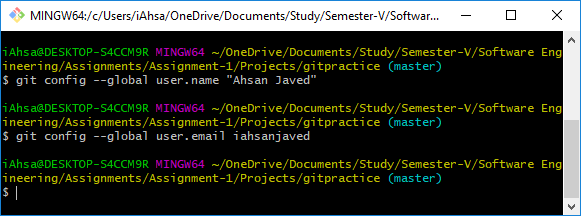
**$ git status**



To set your identity, enter:

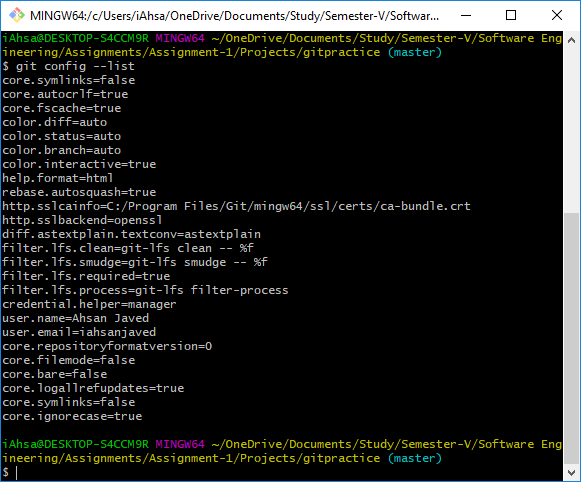
**$ git config --global user.name "Ahsan Javed"**

**$ git config --global user.email iahsanjaved@hotmail.com**



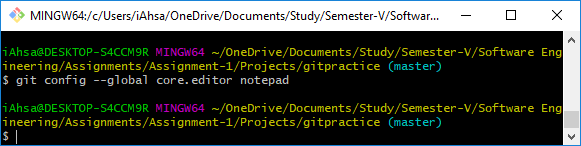
Check values you just set:

**$ git config --list**

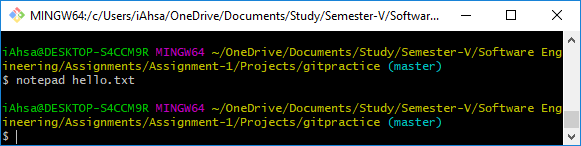


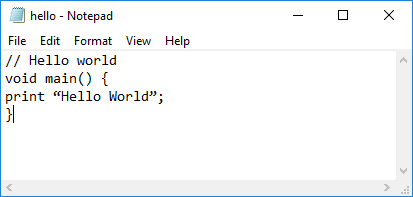
To change the editor to notepad, enter:

**$ git config --global core.editor notepad**



Add a new file to the directory. By typing:  
**$ notepad hello.txt**

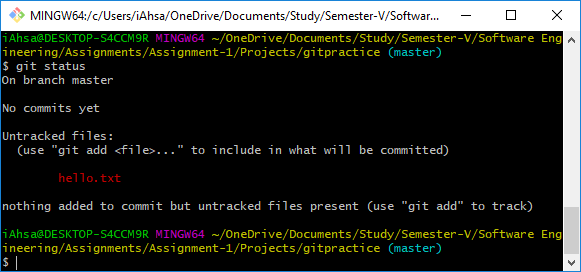




Run git status:

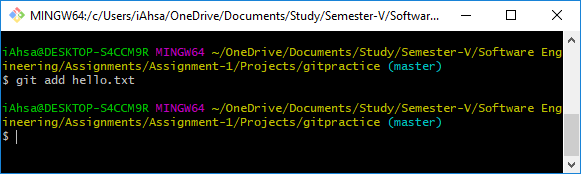
**$ git status**

You should see the file as “untracked”.



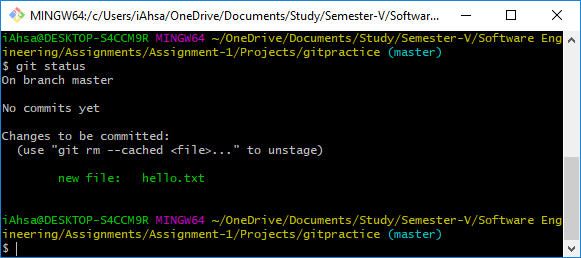
Add the file with:

**$ git add hello.txt**



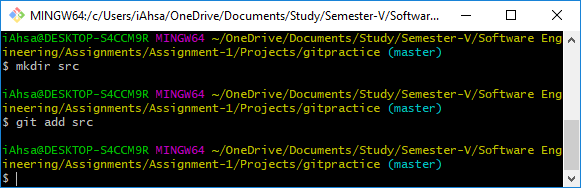
Run git status to verify the file was staged:

**$ git status**



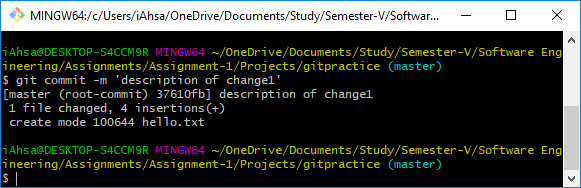
The add command takes the name of a file or directory. If a directory is specified all the contents of the directory and subdirectories are added. For example:

**$ git add src**



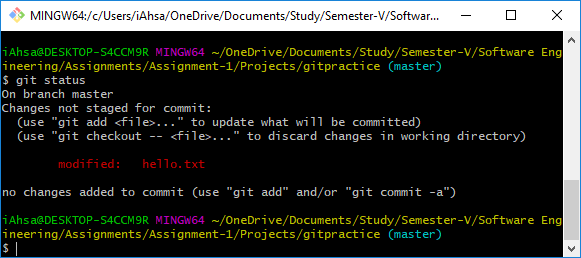
To commit all changes that have been added, enter:

**$ git commit –m ‘description of change1’**



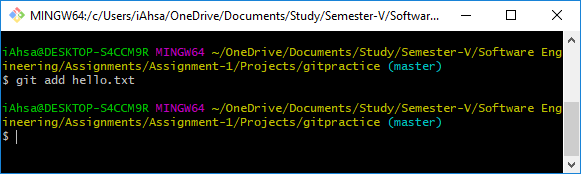
To verify the commit was successful, run git status:

**$ git status**



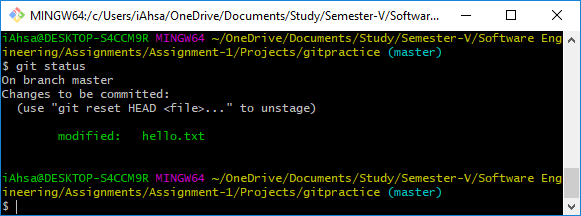
Stage it again with:

**$ git add hello.txt**



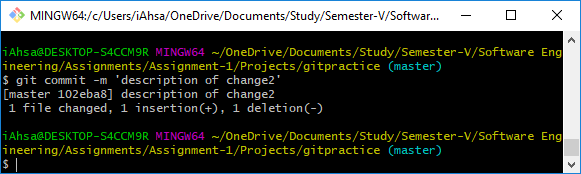
Run git status:

**$ git status**



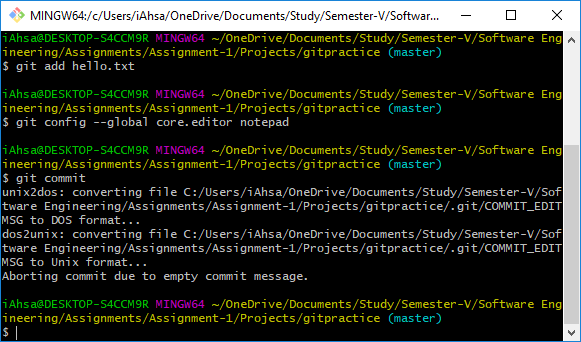
You should see the status of the file as staged but not committed. Now, commit the changes with:

**$ git commit –m ‘description of change2’**



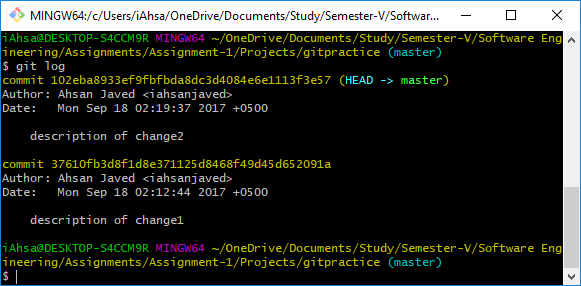
The best way to enter a multi-line comment, is to leave the –m option off the commit. The following will open the default editor for the commit message:

**$ git commit**



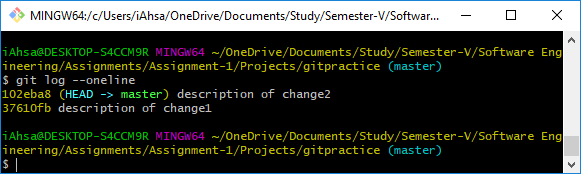
To see a history of changes:

**$ git log**



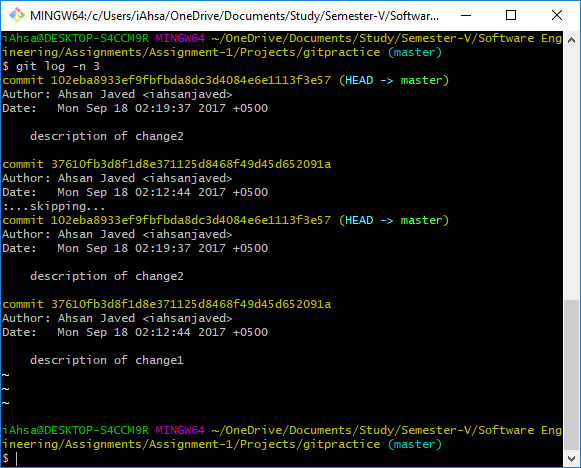
To see an abbreviated history:

**$ git log –oneline**



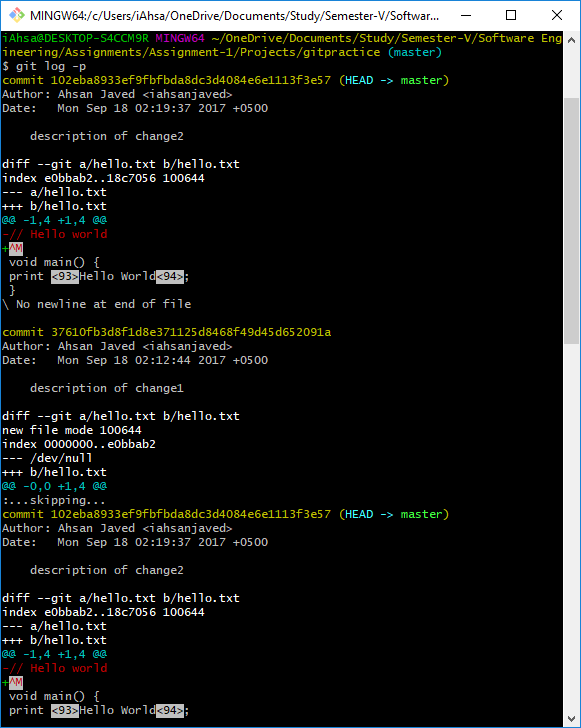
You can also, limit the number of previous commits to show. The following will show the last 3 commits:

**$ git log –n 3**



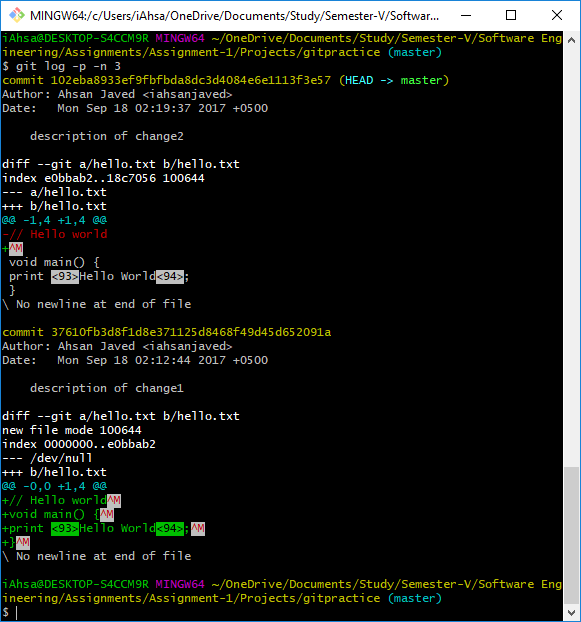
To show the differences introduced in each commit, use the –p option:

**$ git log –p**



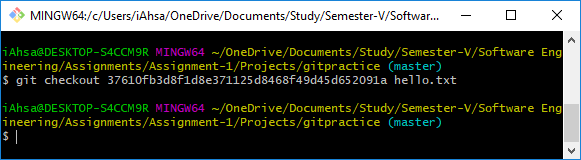
You can of course, combine the two:

**$ git log –p –n 3**



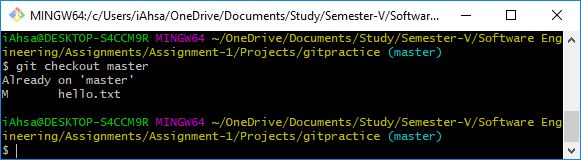
To get back to a previous state:

**$ git checkout** **37610fb3d8f1d8e371125d8468f49d45d652091a hello.txt**



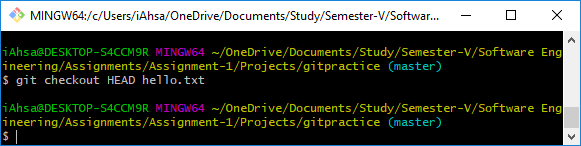
To get back to the latest:

**$ git checkout master**



To get back to the latest version of the file, enter:

**$ git checkout HEAD filename**



A .gitignore file should be committed into your repository (add followed by commit), in order to share the ignore rules with any other users that clone the repository.

Use notepad to create a .gitignore file with the following contents:

// Ignore .class files and everything

// in the bin directory

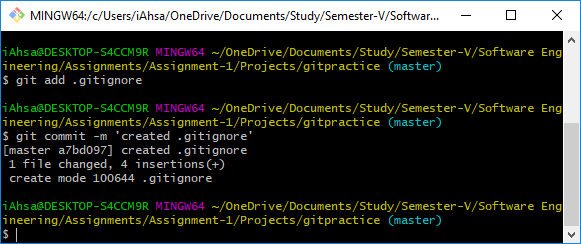
\*.class

bin/

Add and commit the .gitignore file to your project:

**$ git add .gitignore**

**$ git commit –m ‘created .gitignore’**

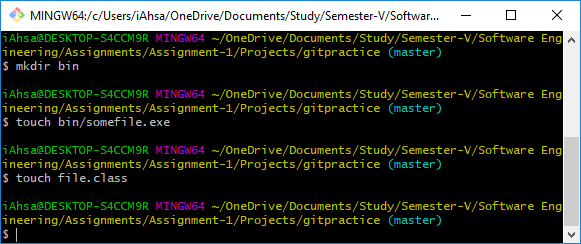


Now, create a bin directory, add a file to it and create a .class file:

**$ mkdir bin**

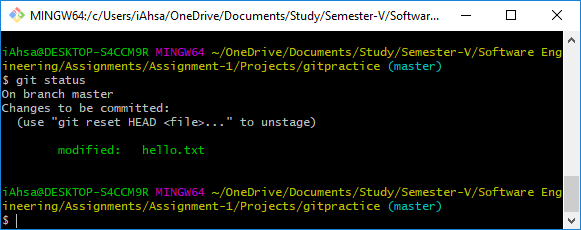
**$ touch bin/somefile.exe**

**$ touch file.class**



Now, check git status and notice these files are ignored (git doesn’t flag these new files as untracked):

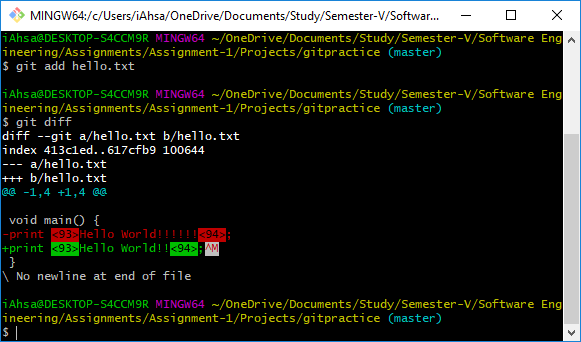
**$ git status**



For more details on what has changed, you need git diff. git diff will tell you (1) what is changed but not staged, and (2) what is staged and about to be committed. Unlike git status, git diff shows exact lines changed/added/removed.

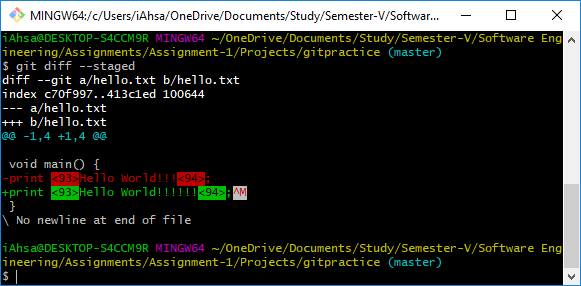
The following shows changed but not added:

**$ git diff**



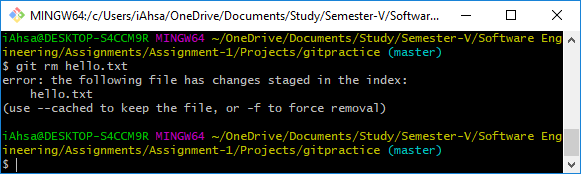
The following shows exactly what will be added in the next commit (exact lines of each file):

**$ git diff –staged**

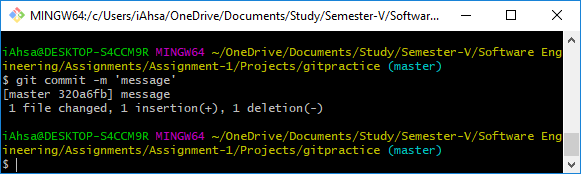


To delete a file from your project and remove it from being tracked, you can’t just delete and commit. To delete a file enter:

**$ git rm hello.txt**

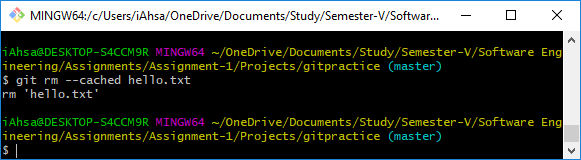


**$ git commit –m ‘message’**



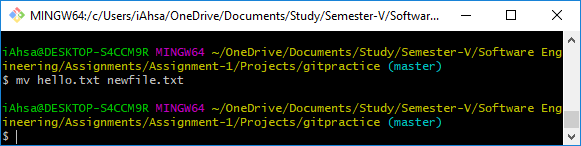
git rm removes the file from the directory and causes git to stop tracking it. If you just want git to stop tracking it, use:

**$ git rm –cached hello.txt**

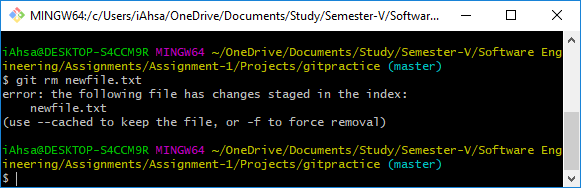


If you want to rename a file, one option is to rename it outside of git and then delete the old one and add the new one in git:

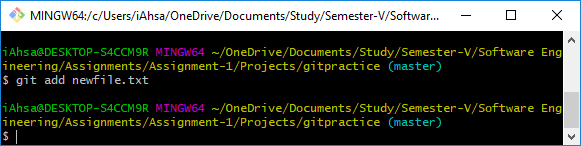
**$ mv hello.txt newfile.txt**



**$ git rm oldfile.txt**

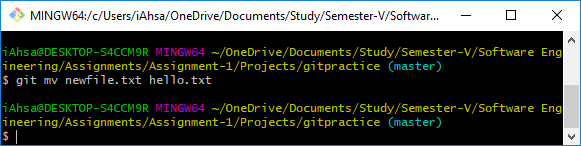


**$ git add newfile.txt**



Or, you can accomplish the same using the git mv command:

**$ git mv newfile.txt hello.txt**

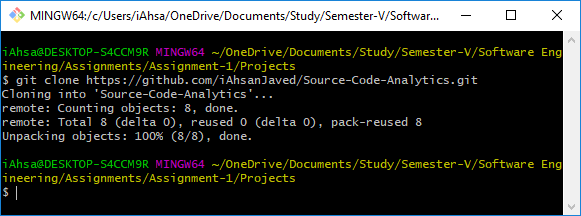


# EXERCISE-1: Cloning an existing repository

Oftentimes the git repository will already exist on a remote server. The common scenario is you join a project and want to modify existing documents or upload new ones. In cases like this, you (1) create a local copy or clone of the repository, (2) add/modify existing documents. (3) push your changes back to the central or upstream repository.

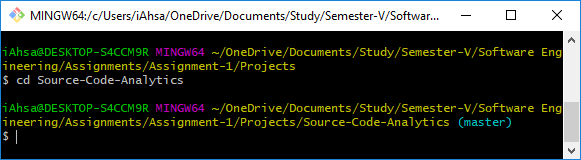
Clone the remote repository with:

**$ git clone** [**https://github.com/iAhsanJaved/Source-Code-Analytics.git**](https://github.com/iAhsanJaved/Source-Code-Analytics.git)



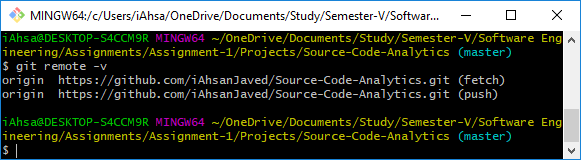
The command above will create a new subdirectory gitpractice and initialize it with the remote repository. cd (change directory) into the new directory:

**$ cd Source-Code-Analytics**



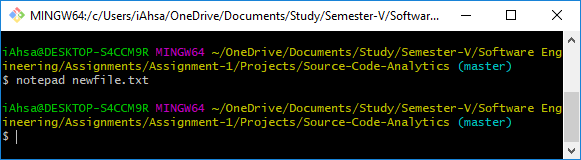
Verify it the clone command created remote references to the remote repository:

**$ git remote –v**

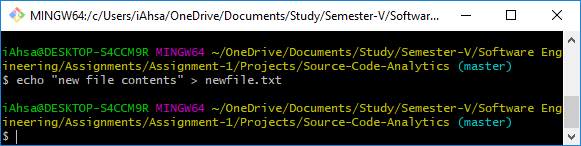


Modify one of the downloaded files or add a new file to the directory:

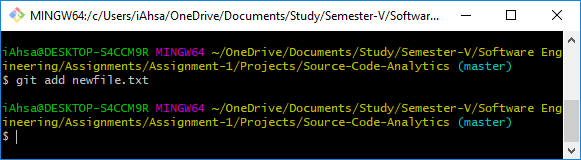
**$ notepad newfile.txt**



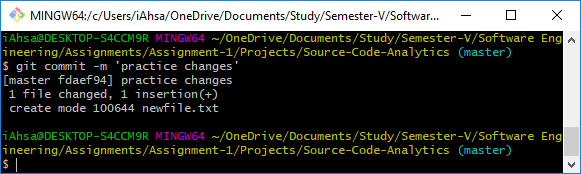
**$ echo “new file contents” > newfile.txt**



**$ git add newfile.txt**

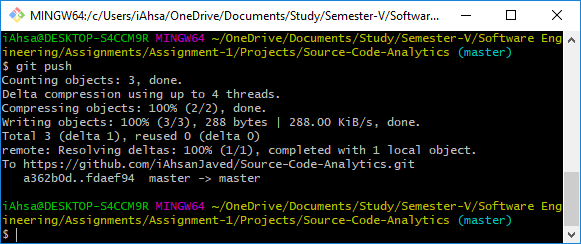


**$ git commit –m ‘practice changes’**

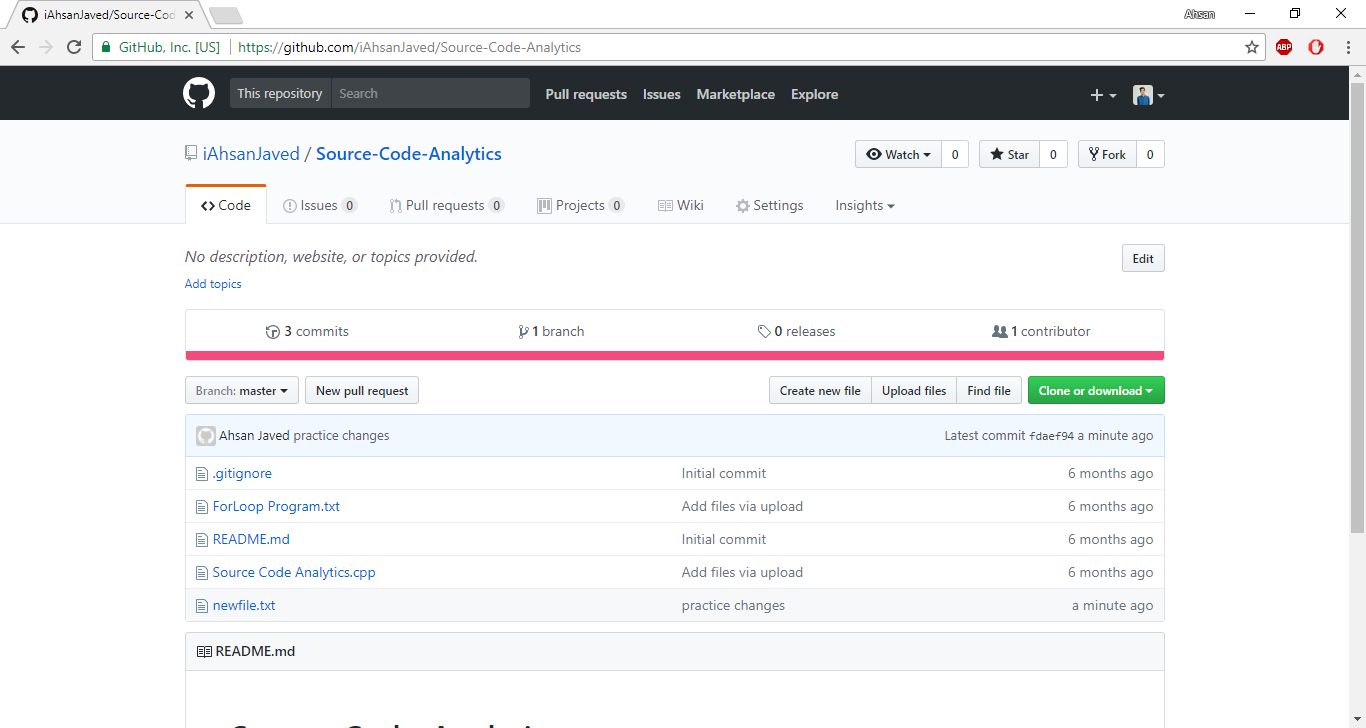


To push your changes to the remote server:

**$ git push**

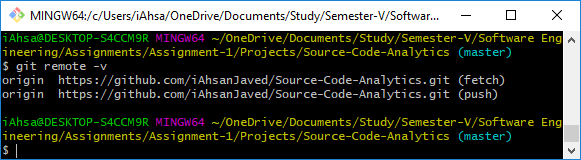


Browse github.com to verify new changes were uploaded correctly. When you clone a repository with git clone, it automatically creates a remote connection called origin pointing back to the cloned repository.



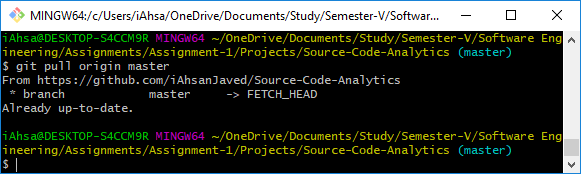
You can verify this with the following command:

**$ git remote -v**



Pro tip! As time goes on others may push their own changes to the central repository. On a regular basis you should update your local copy with changes made on the central repository:

**$ git pull origin master**



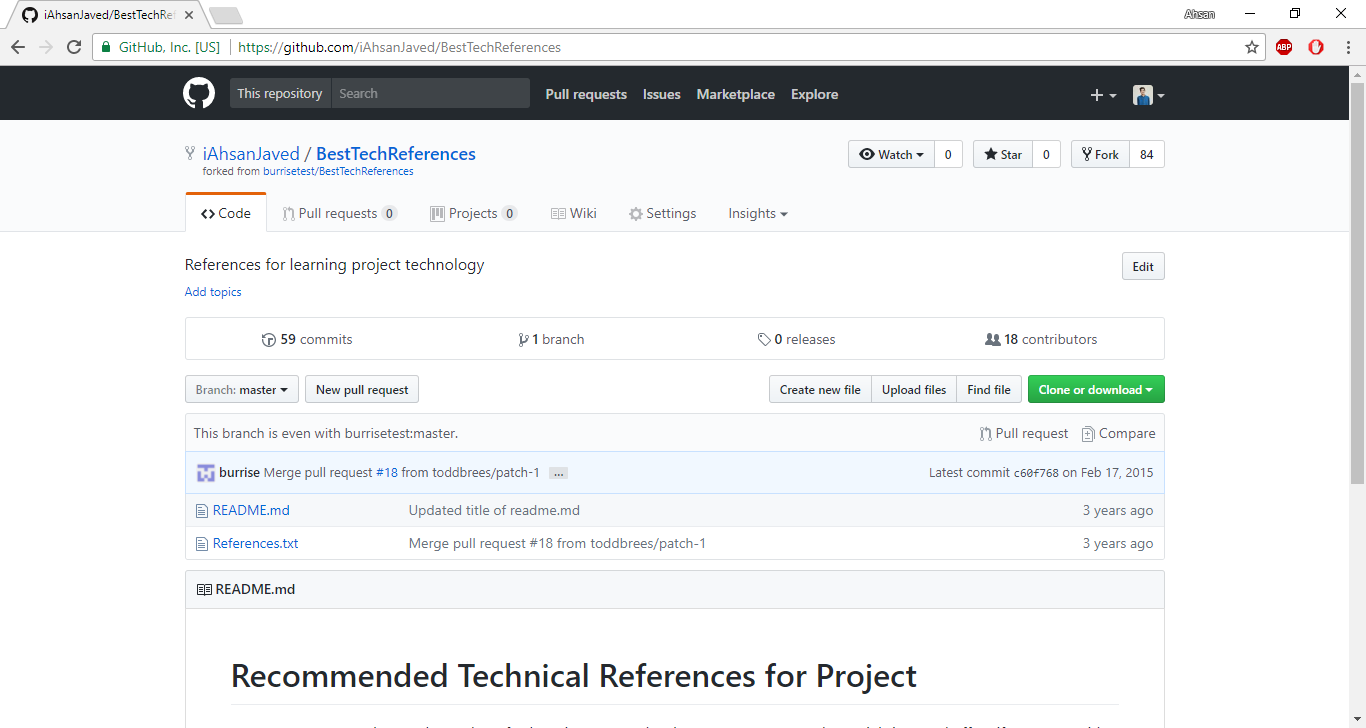
# EXERCISE-3: Contributing to an existing github project

This exercise will give you practice with contributing to an existing github.com project. There are two options for contributing to an existing github project. First, the owner of the project could give you read/write access (not very likely), or you could use the fork & pull request method. This exercise uses the fork & pull request method.

In this exercise you will fork a repository, commit and sync changes to your forked copy on github and then issuing a pull request to the owner of the original project to have your changes merged with the original.

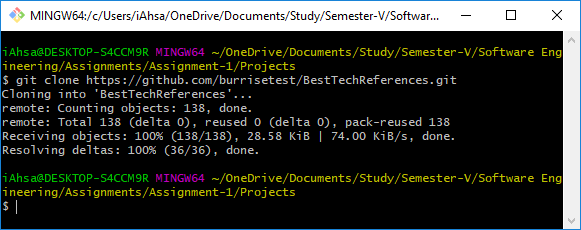
First, fork the following repository: https://github.com/burrisetest/BestTechReferences.git. You will find it under the user ID: burrise. (Hint: log on to github.com, find the repository, press “Fork”.)

When you fork a github repository, a copy of the repository appears under your account at github.

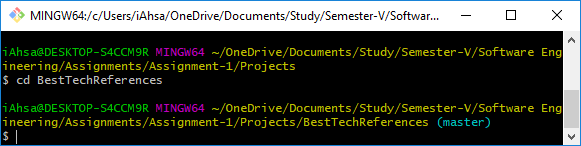


Clone the forked repository to a local computer. (See exercise #2 above.) (Note, you will have to change burrisetest to your github user ID.)

**$ git clone https://github.com/burrisetest/BestTechReferences.git**

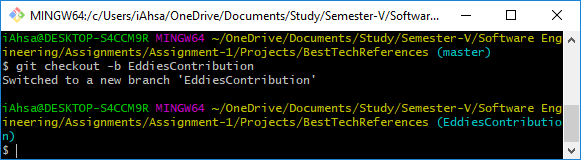


**$ cd BestTechReferences**



Create a topic branch for performing work.

**$ git checkout -b EddiesContribution**



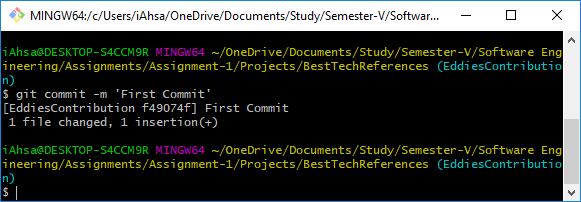
(Or:

**$ git branch EddiesContribution**

**$ git checkout EddiesContribution**

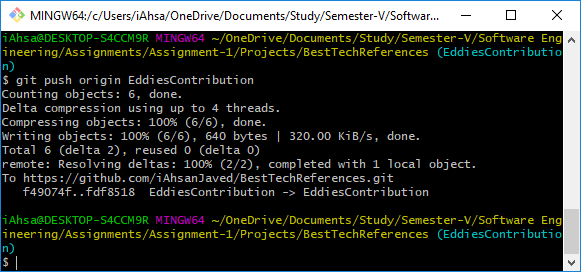
)

Open the local file References.txt and add a reference. Be sure to include your name so we know who to credit. Once you are done with all edits and commits, commit your changes to your topic branch.



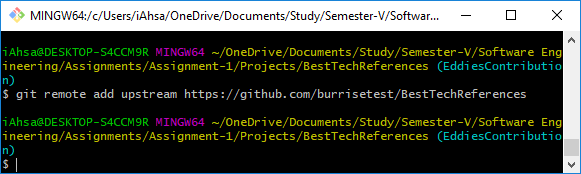
You could do a git push origin EddiesContribution at this point:

**$ git push origin EddiesContribution**



To make it easier for the owner to merge your changes, you should add an upstream remote and pull in any upstream changes there are. The following command will add an upstream remote:

**$ git remote add upstream** [**https://github.com/burrise/BestTechReferences**](https://github.com/burrise/BestTechReferences)



Now, pull in upstream changes:

Option 1:

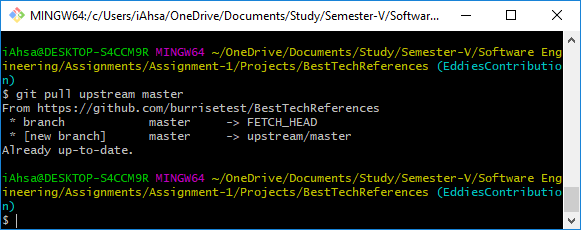
**$ git fetch upstream**

**$ git merge upstream/master**

(Note, you may want to use the --no-ff flag. It will always generate a merge commit: git merge -- no-ff upstream/master.)

Option 2:

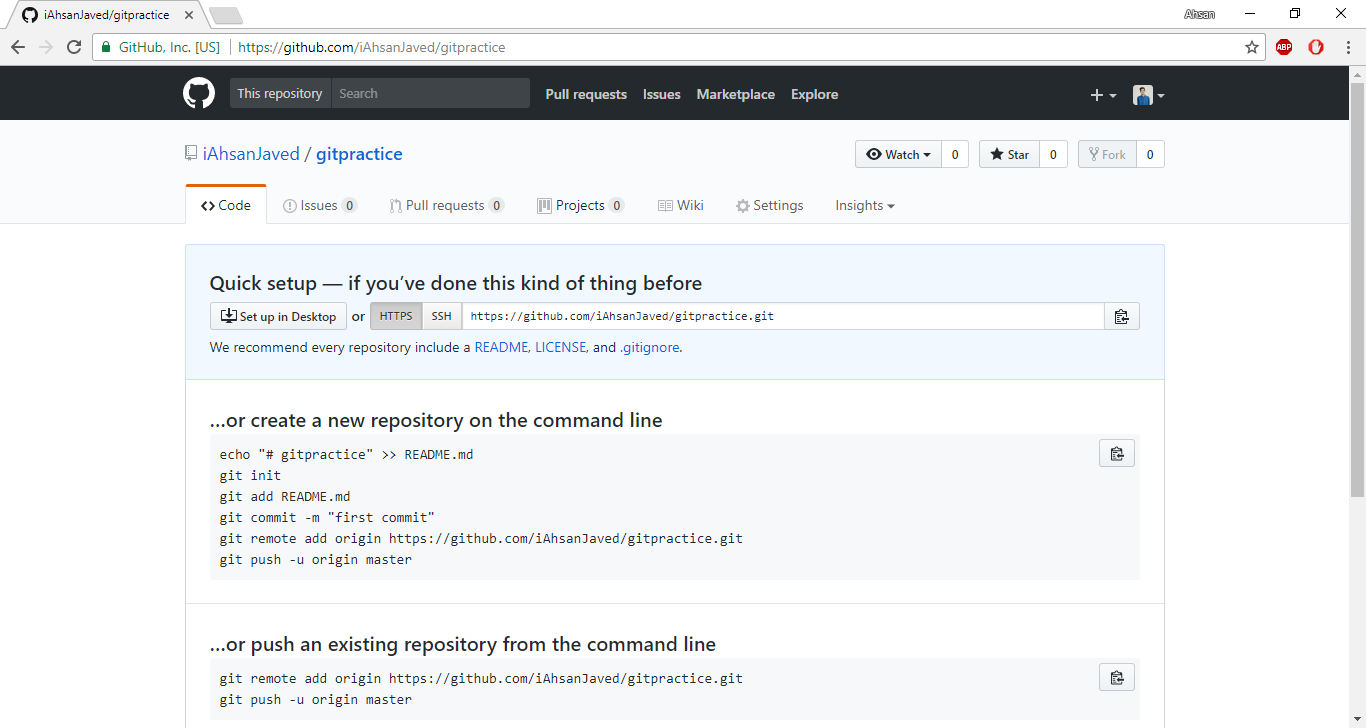
**$ git pull upstream master**



If there are no upstream changes since you forked, there will be nothing to merge. If there are upstream changes but they don’t overlap with the changes you made, the changes will be merged automatically.

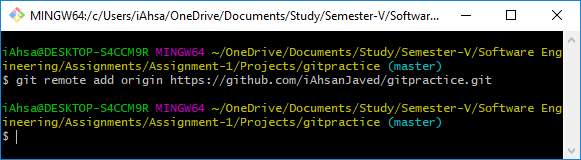
# EXERCISE-4: Adding a remote

To create a remote shared repository, go to github.com and create a new repository.



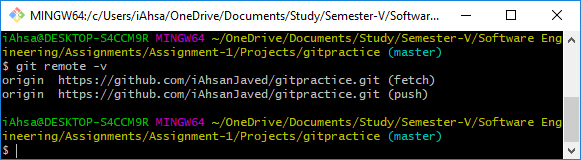
Add it as a remote:

**$ git remote add origin https://github.com/burrise/gitpractice.git**



Issue the remote command to verify it was added successfully:

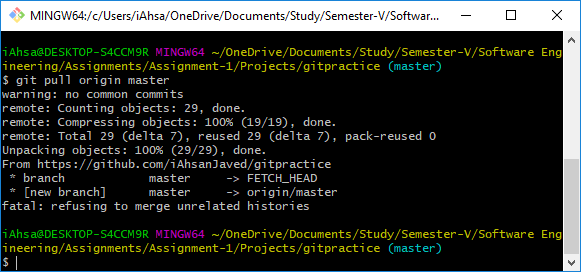
**$ git remote -v**



Your local repository is out of sync with the remote that was just created. They have to be in sync before you can push your local changes.

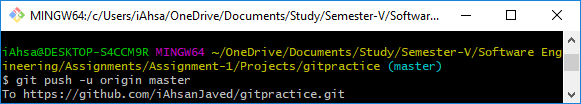
The following command will merge the contents of the remote repository (just a readme file) with your existing local repository:

**$ git pull origin master**



Push your changes with:

**$ git push -u origin master**



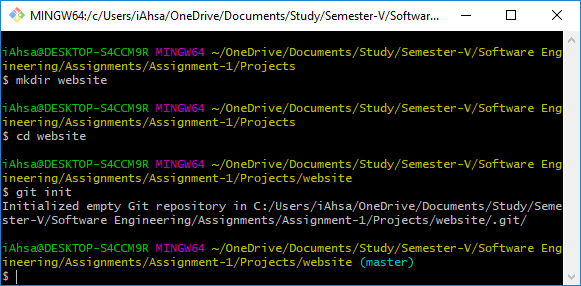
# EXERCISE-5: Branching and merging

Create a new directory and init a git repository:

**$ mkdir website**

**$ cd website**

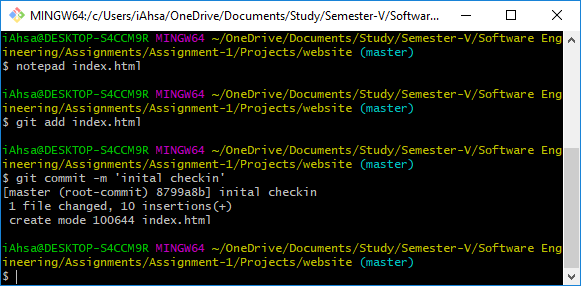
**$ git init**



**$ notepad index.html**

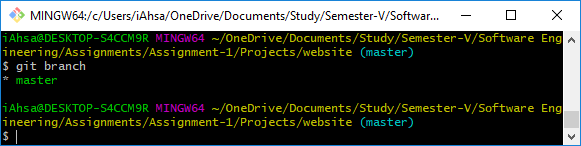
**$ git add index.html**

**$ git commit –m ‘initial checkin’**



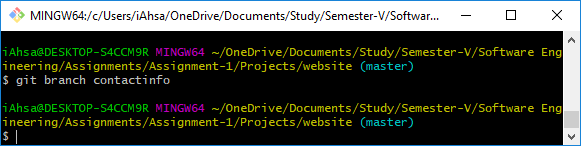
To see, run the command that lists all branches:

**$ git branch**



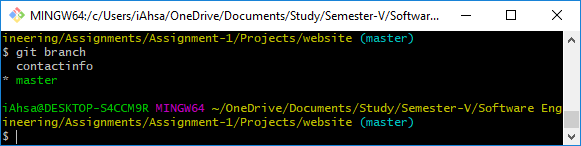
To create a branch for the new feature we will be adding:

**$ git branch contactinfo**



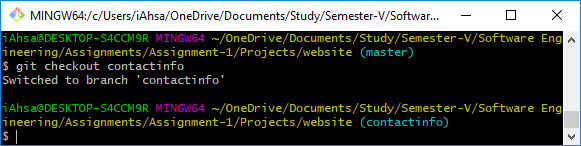
List the branches again:

**$ git branch**



You have a new branch but are still on the master branch. To switch to the contactinfo branch:

**$ git checkout contactinfo**

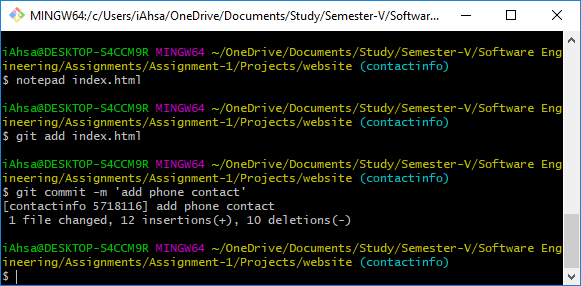


Now, edit the file to add contact info:

**$ notepad index.html**

**$ git add index.html**

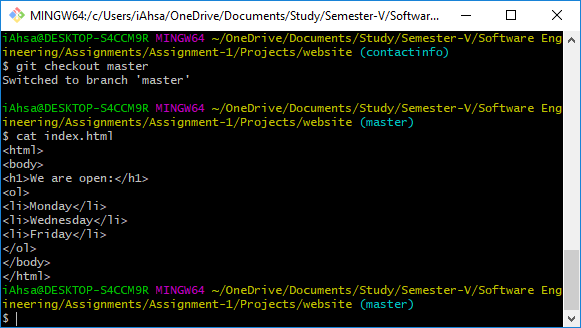
**$ git commit –m ‘add phone contact’**



If you switch back to master, the files in your directory will appear as they were before you created a branch. Try it:

**$ git checkout master**

**$ cat index.html**



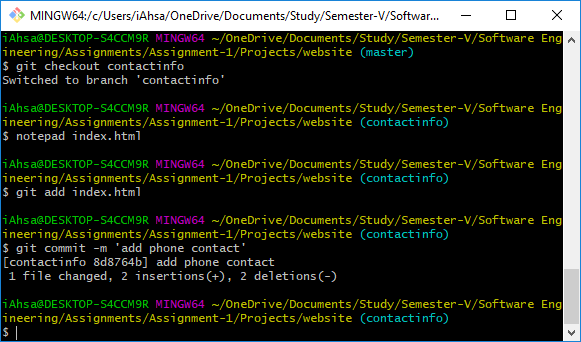
Now, switch back to the contactinfo branch and finish the feature:

**$ git checkout contactinfo**

**$ notepad index.html**

**$ git add index.html**

**$ git commit –m ‘add phone contact’**



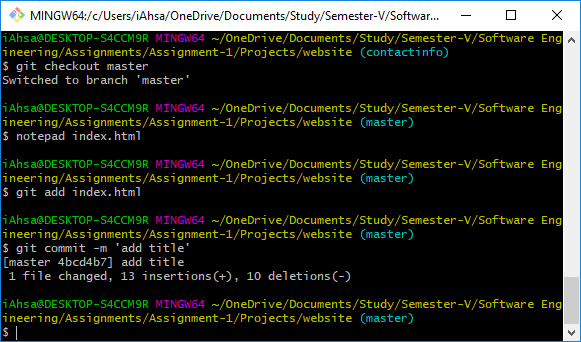
For a more realistic scenario, let’s switch back to the master branch and make some changes:

**$ git checkout master**

**$ notepad index.html**

**$ git add index.html**

**$ git commit –m ‘add title’**

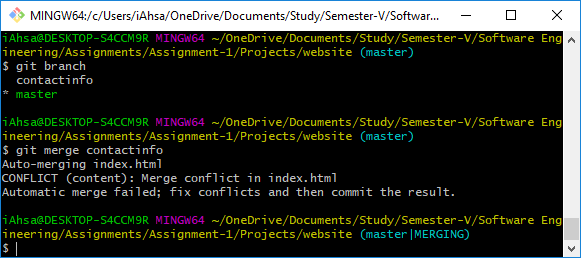


Let’s merge the two. You always merge into a branch. In other words, you start from the branch you want to merge into. Check to make sure you are on the master branch:

**$ git branch**

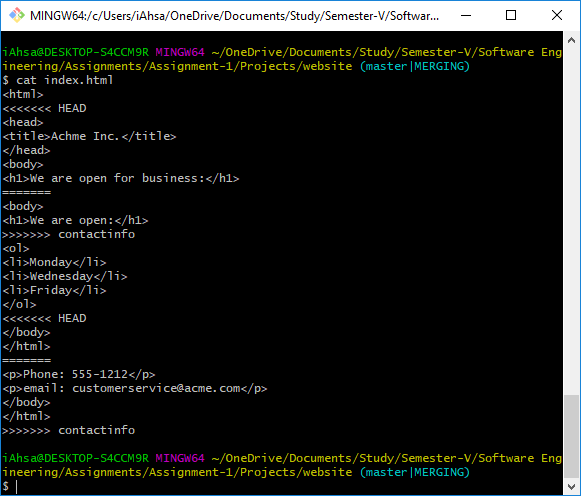
To merge the contactinfo branch:

**$ git merge contactinfo**



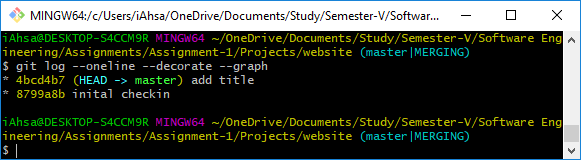
List the contents of index.html to verify the merge was performed correctly:

**$ cat index.html**



Use the following command to see a visual representation of the commits:

**$ git log --oneline --decorate –graph**



We are done with the contactinfo branch so it can be deleted:

**$ git branch -D contactinfo**

