Code Repository Tools

A source code repository is a file archive and web hosting facility where a large amount of source code, for software or for web pages, is kept, either publicly or privately. They are often used by open-source software projects and other multi-developer projects to handle various versions.

There are different tools available on the internet, few of them are :

* Subversion
* Git
* Mercurial
* Perforce

Git and Mercurial are distributed source controls tools, whereas SVN and Perforce are not.

Git is immensely faster and more convenient than SVN, CVS, and TFS. It is fast

becoming the defacto standard mainly because there are no tools as powerful and

versatile as it is; it is a wonderful masterpiece of code. Behind the success of Git lies

much more: it is a revolutionary approach to writing code.

# Features of Git

1. **Setting up a new Repository**

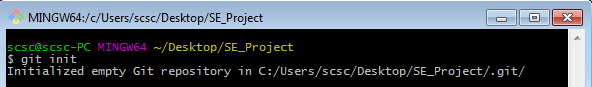
The first step is to set up a new repository (or repo, for short). A **repo** is a container

for your entire project; every file or subfolder within it belongs to that repository,

in a consistent manner. Physically, a repository is nothing other than a folder that

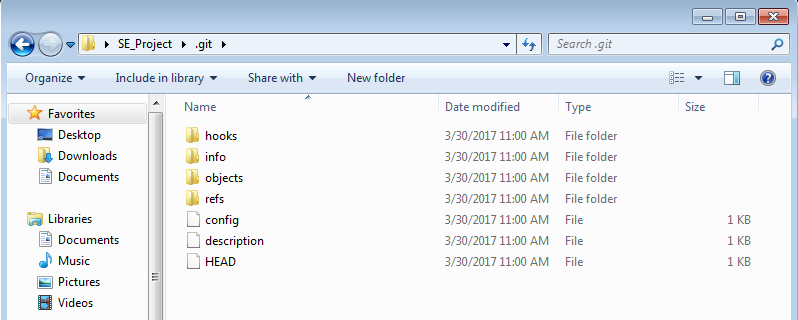
contains a special .git folder

Typing ***git init*** command initializes the repo for our code.



Git created a .git subfolder. The subfolder (normally

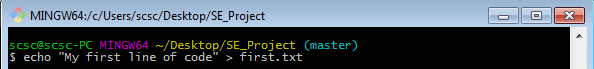
hidden in Windows) contains some files and folders, as shown in the next screenshot:



Now that we have a repo, we can start to put files inside it.

1. **Adding a File**

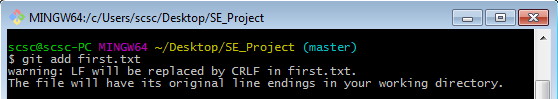
Lets create a text file and start our coding in it.



* ***echo*** is the same as creating a new file then editing it with notepad

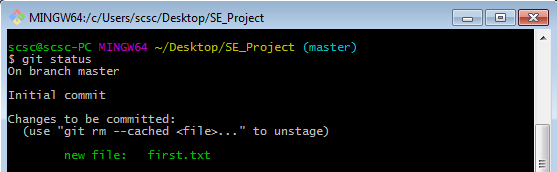
Now that we have added a file, we have to tell Git to add this file in our repository and to start it’s tracing.

***git add first.txt*** command add’s it to our repository



Using the git status command, we can check the status of the repo, as shown in

the following screenshot:



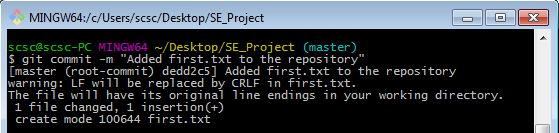
As we can see, git has added our file to the repository and is tracking it’s status. Now we need to commit our changes.

1. **Commit the added file**

At this point, Git knows about MyFile.txt, but we have to perform another step

to fix the snapshot of its content. We have to commit it using the appropriate ***git***

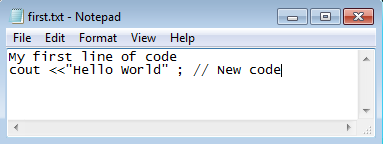
***commit*** command.



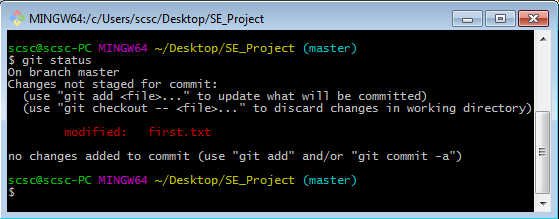
Now that we have commited our changes, we can safely change the contents of the first.txt file without having to worry about the errors (as we can go back to the previous commit and recover our code to till that commit). Notice the –m flag here, it is for giving the appropriate message to our commit, so that we could identify the code or point when commiting back to the previous codes.

1. **Modify a commited file**

Now let’s try to modify a commited file and see how to deal with it

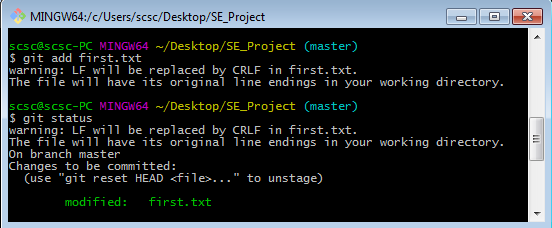


Now let’s check the status of our repository by typing ***git status***



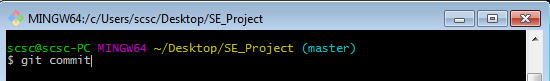
We can see that Git has noticed the modifications and has marked our file as modified. It gives us the option to save the changes (by adding them to commit) or to discard the changes (we will try this option later).

Since this code is valid, let’s try to save the changes and add this to our commit.

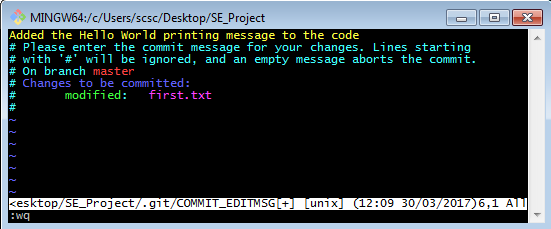


Now that we have added the file, Git marked our code ready and it will be saved when execute the commit command.

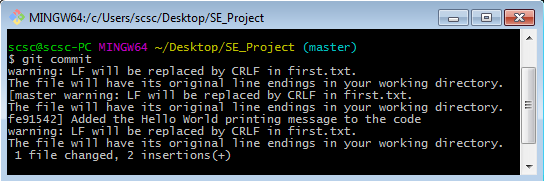
Let’s commit our code. But this time, we will not add the message flag to our commit and see how the Git react’s to this :



Git opened up it’s prefered text editor **Vim** (Vi Improved). You can configure Git to use your own preferred editor, but if you don't do it, this is what you have to deal with. Vim is powerful, but it has a strange way of dealing with text, so to start typing, you have to press i for inserting text, as shown in the following screenshot:



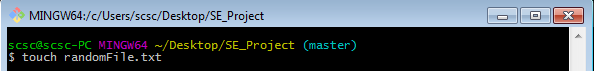
After entering your commit message, you can come out of the editing mode by pressing ***Esc*** key, then you can save your commit by :wq (w for writing the changes and q to exit the editor)



1. **Unstaging a file**

Staging area or index is a virtual place where the files you want to include in the next commit are placed. The files can be staged using the ***git add*** command. So what if a wrong file is added into the index and we want to remove it so that it should not be added in our next commit?

Let’s add a file into the staging area first and then remove it from the index.

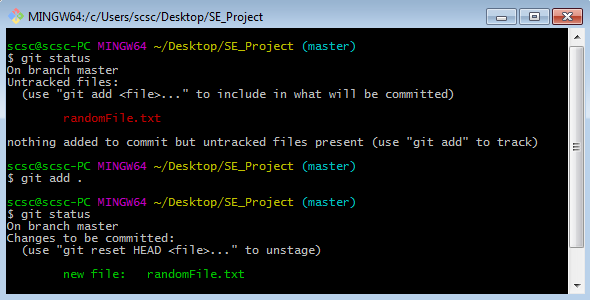


* ***touch*** is the same as creating a new file from windows GUI

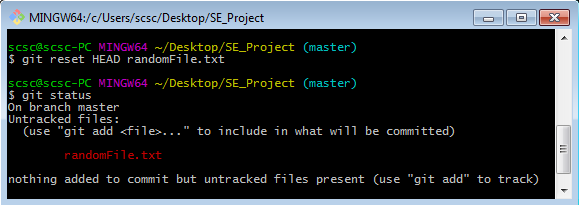
Check the status of repository (***git status***)

Then add the new file into the staging area (***git add .***)

* git add . ( . means that all the untracked files will be added into the staging area)



Now that the ‘randomFile.txt’ has been added into the index, we can remove it by the following command :  
  
***git reset HEAD <file>***

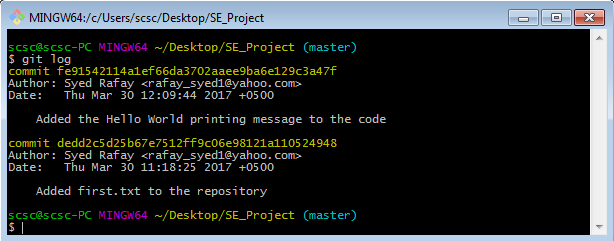


Notice that the reset command remove the file from the staging area and it’s status is now set back to ‘untracked’. Git is now not concerned with what happens with that file as it is not tracking this file now. We can safely delete this unwanted file now.

* Note that ***HEAD*** pointer is the reference to the last commit we did.

1. **Viewing the History**

We can check the history of our repository by the ***git log*** command



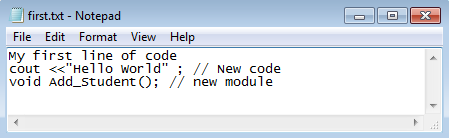
It shows the commit id (40 characters long, which is unique for every commit), Author of the commit, Date and Message of that commit.

We can set the author name and email by the following commands :

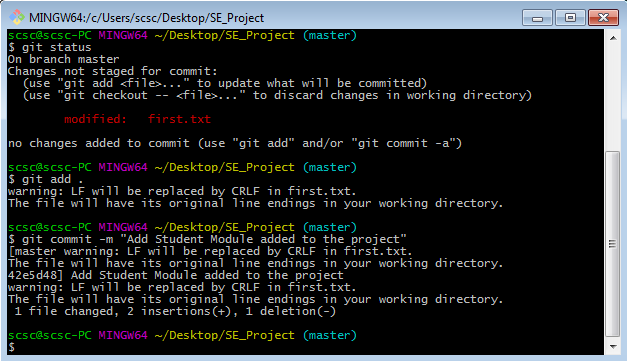
* ***git config user.name*** **<***user-name***>**
* ***git config user.email*** **<***users-email***>**

1. **Switching back to previous commits**

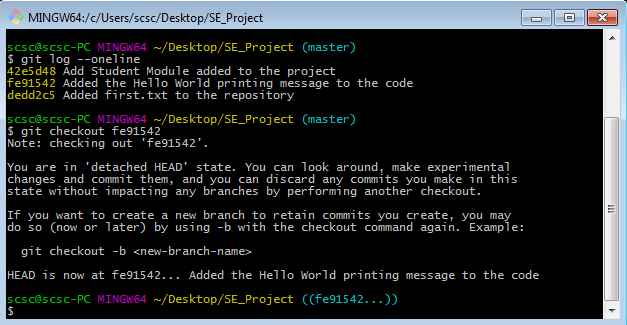
We can also switch back to the previous commits if something in our code goes wrong. Let’s also try that out.



For example, we added the Add Student module and commited the code. But it made our code erroneous.



Now let’s switch back to the commit where Add Student module wasn’t introduced.

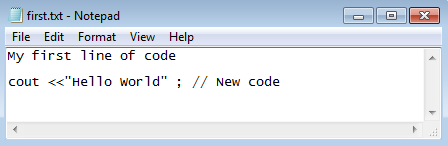


Let’s check the commit history by ***git log --oneline***

* ***--oneline*** flag is used with above command to shrink the logs to one line

We switched back to previous commit by ***git checkout*** **<***commit-id***>**

Let’s check the contents of first.txt file now :



As we can see, the contents are switched back to the previous state and the erroneous code has been removed (the HEAD pointer is now at the previous commit)

The following command is used to proceed forward to the commit from where we came from :

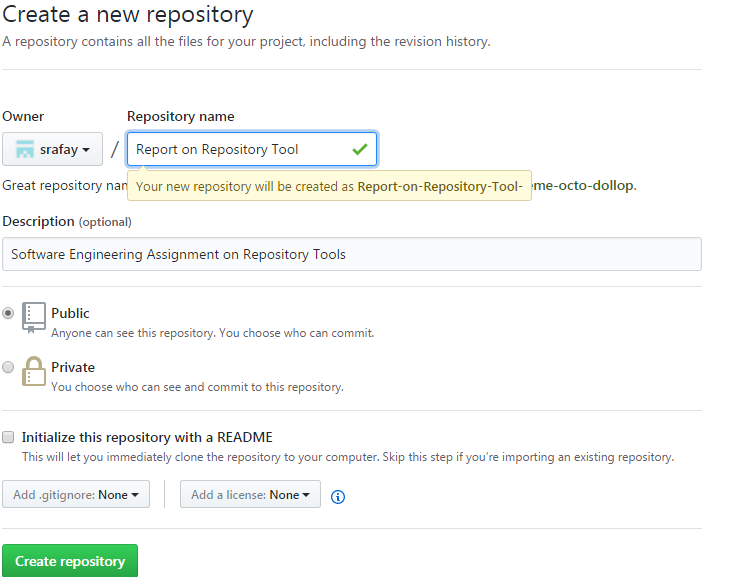
* ***git checkout –***

1. **Working Remotely**

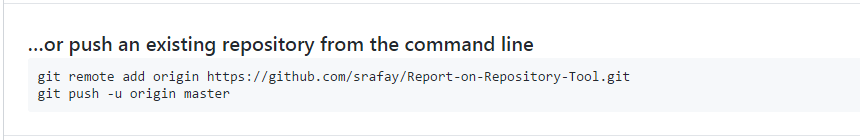
Now that we have a good understanding of git in the offline mode, we can upload our project online and share it with our colleagues or other developers. One of the most common and free to use remote is GitHub. First we need to create an account at <https://github.com>

After creating an account, goto “Repositories” tab and click on Green colored button “New”.

Choose the Repository name and enter the description of your project. Click on Create Repository and now we are ready to push our project to the remote server.

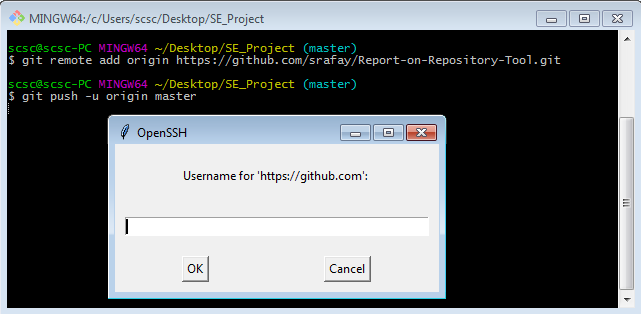


Read the instructions for pushing an existing repository to the server



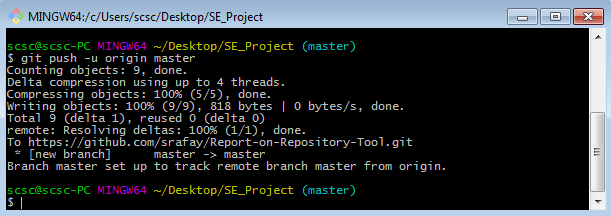
Copy these commands and paste them into git bash command line.

After you enter these commands, it prompts for username and password

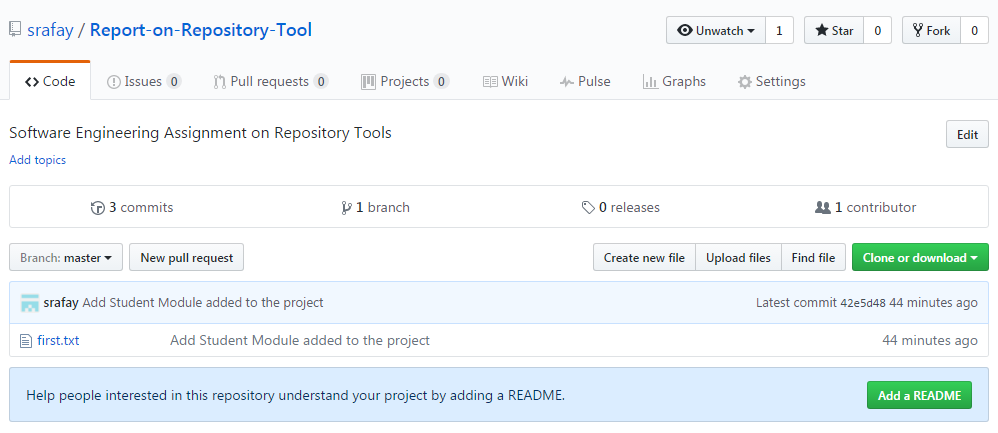


The repository is uploaded to the GitHub remote server after you enter your credentials.

* ***-u*** flag with the push command is to keep our local branch in sync with the remote branch (branches are not covered in this report)
* ***origin*** is the default remote name of our repository.



We can see that the repository is now available at the GitHub



We can make changes to our project, add files and commit them. We can then upload our updated work by the command :

* ***git push***