**SWE 525 GIT Version Control Mid Term Exam 04/10/2016**

**ITU Student Name: Kalya Archana Bhat**

**ITU Student ID: 87271**

1. **List out the key difference between a centralized version control system and distributed version control system.**

**CVCS – Centralized Version Control System**

* Centralized version control systems are based on the idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy.
* Committing” a change simply means recording the change in the central system. Other programmers can then see this change. They can also pull down the change, and the version control tool will automatically update the contents of any files that were changed.
* Most modern version control systems deal with “changesets,” which simply are a groups of changes (possibly to many files) that should be treated as a cohesive whole

**DVCS- Distributed Version Control System**

* DVCS do not necessarily rely on a central server to store all the versions of a project’s files. Instead, every developer “clones” a copy of a repository and has the full history of the project on their own hard drive. This copy (or “clone”) has all of the metadata of the original.
* Performing actions other than pushing and pulling change sets is extremely fast in DVCS, because the tool only needs to access the hard drive, not a remote server.
* Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once.
* Everything but pushing and pulling can be done without an Internet connection. So you can work on a plane, and you won’t be forced to commit several bug fixes as one big changeset.
* Since each programmer has a full copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback before showing the changes to everyone.

List down any two centralized version control system and 2 distributed version control system.

**Two centralized version control**

* **Subversion**
* **Perforce**

**Two Distributed version control**

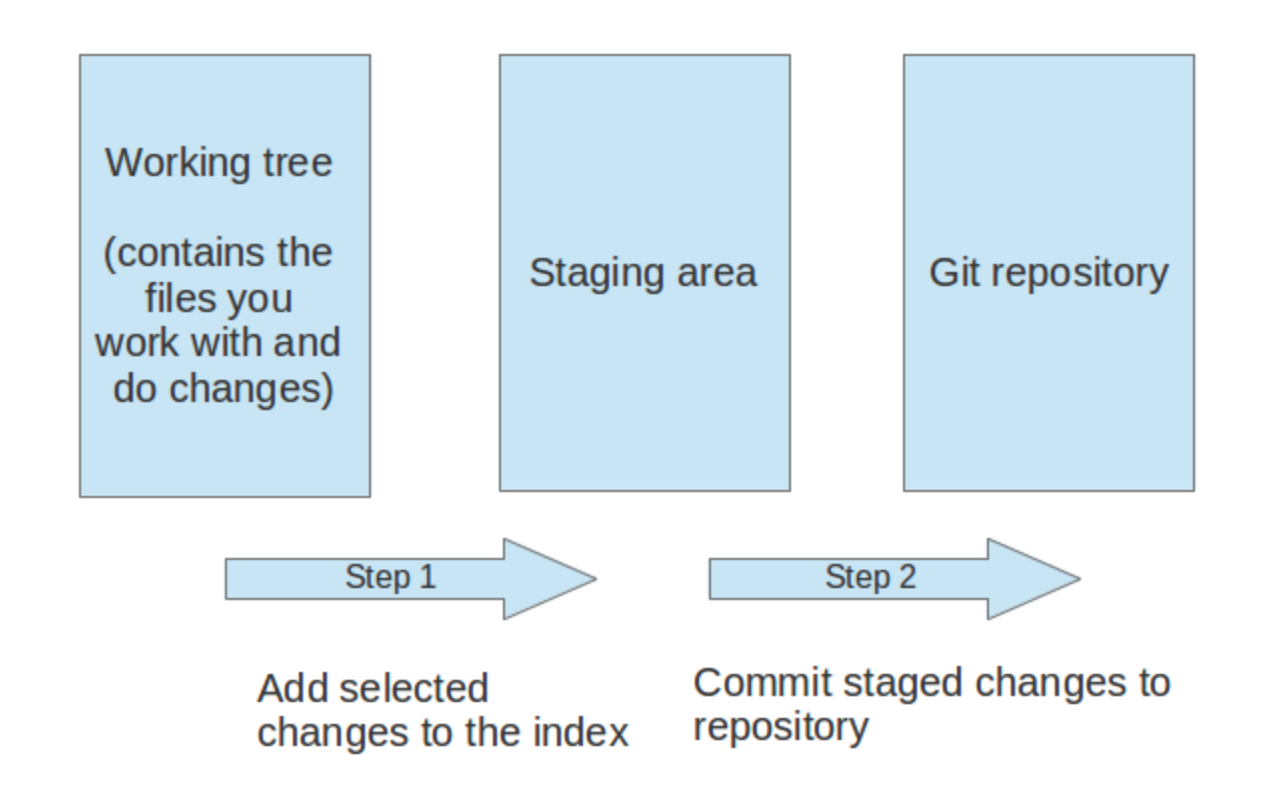
* **Git**
* **Mercurial**
* **Bazaar**
* **Darcs**

What are the advantages of git VCS over other VCS.

**Advantage of git VCS over other VCS**

* Git is decentralized. Means everything can be done without internet. Imagine you are a developer on the road, you develop on your laptop and you want to have source control so that you can go back 3 hours.
* GIT stores content as metadata.
* Frictionless Context Switching.
* Git has a nice feature to create “patches.”  They are simply changes to code, very similar to a diff.  The idea is that you create patches from commits you’ve only made on your local copy of the repository.
* With Git, deleting an abandoned branch is simple and clean

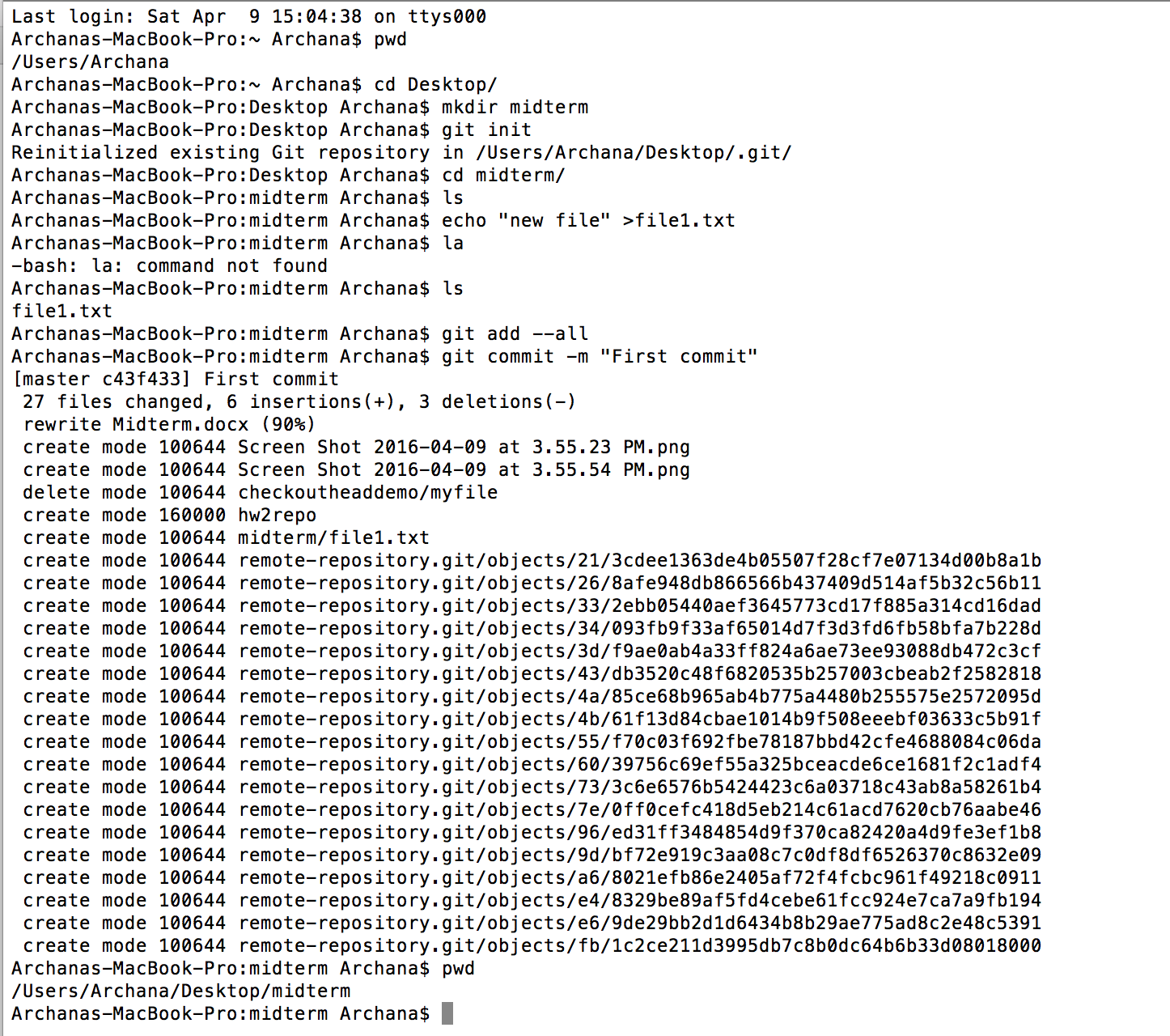
What are the different states of a file in the Git VCS.



Different states of file in the Git VCS are

* Working tree
* Staging Area
* Git repository

1. Perform following Remote Repository operations. Please put commands, screenshots and other information where needed. More detailed oriented response will get higher points. Give me your github repo link also with following steps added.
2. Create a repository on Github and configure your local repo to point to the remote repository

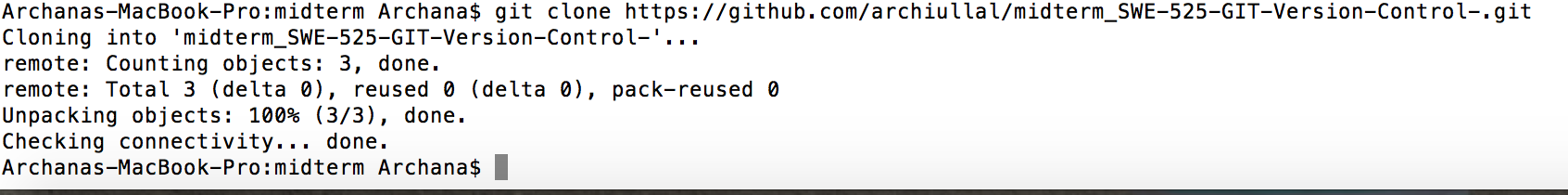


So I have created repo in my local disk called **midterm.**

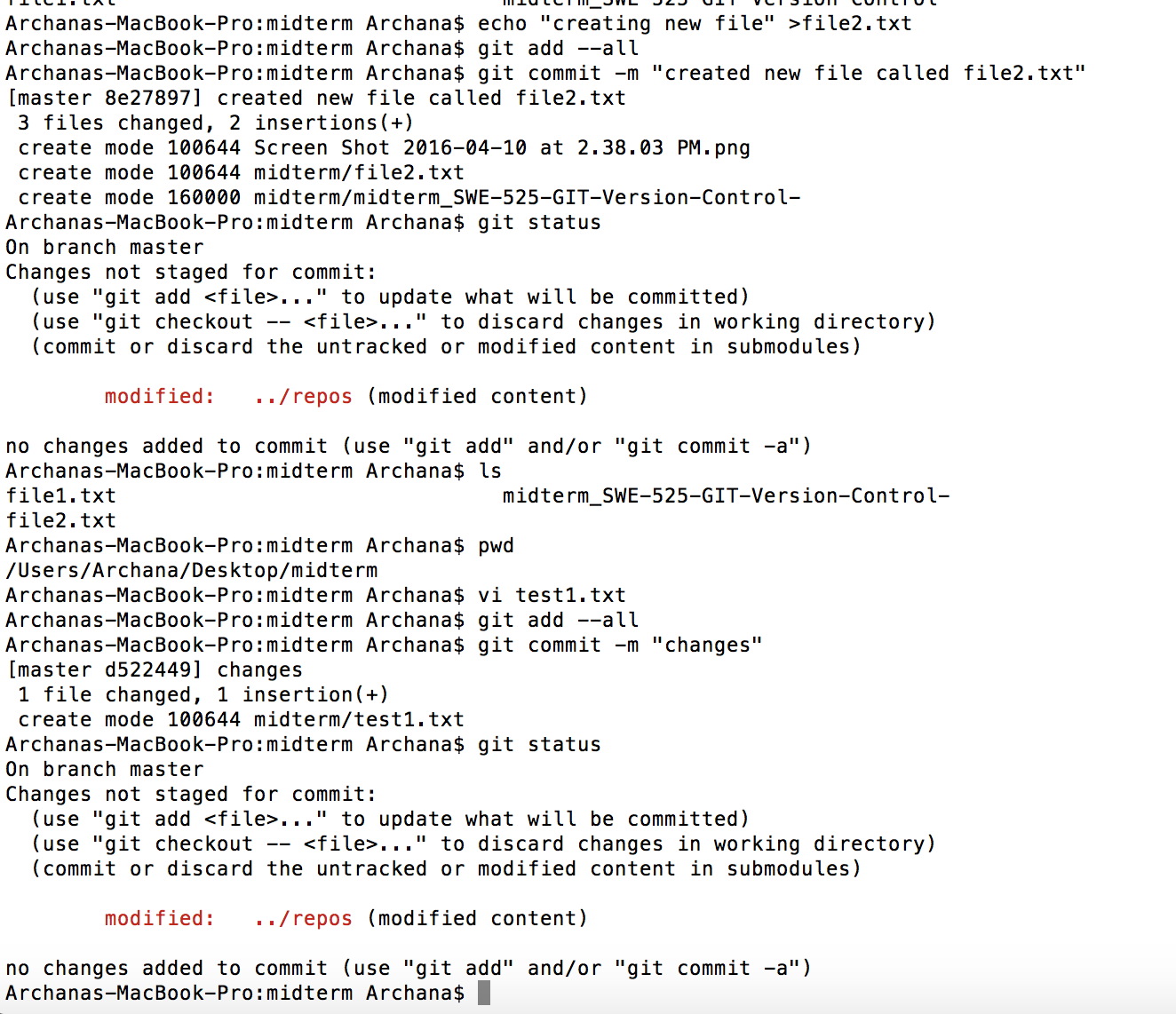
Created repo in **GITHUB called** midterm\_SWE-525-GIT-Version-Control

The clone it.

git clone <https://github.com/archiullal/midterm_SWE-525-GIT-Version-Control-.git>

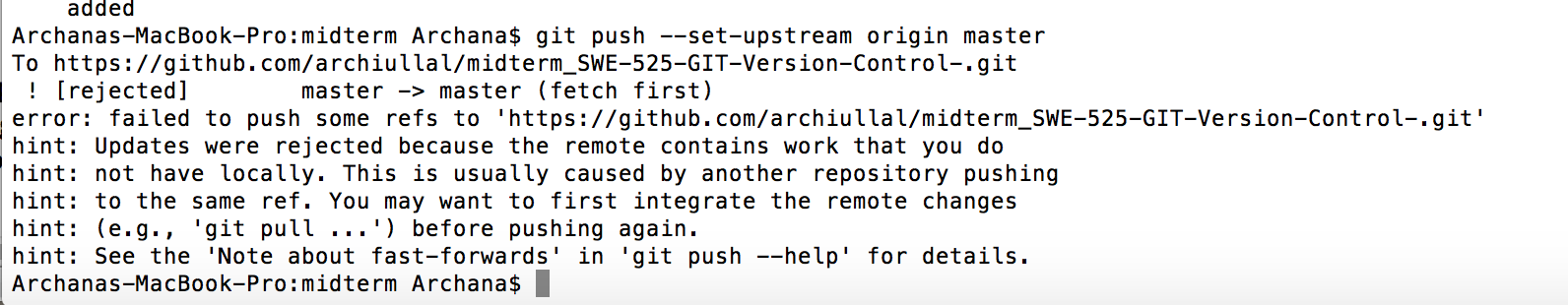


1. Perform some operation like add, remove, modify and finally push your changes to the remote repository

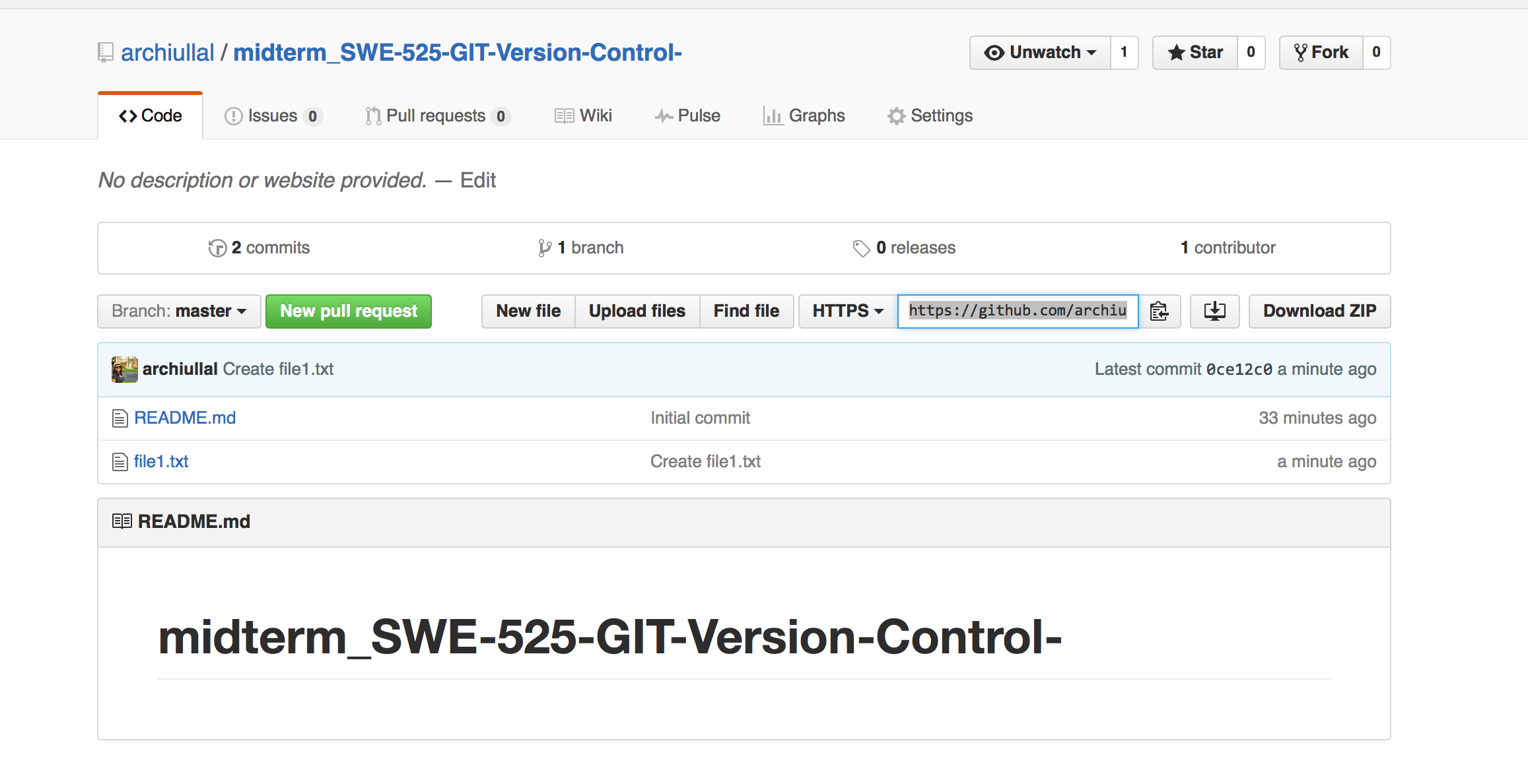


created some new files , made some changes.

Then I pushed my changes.

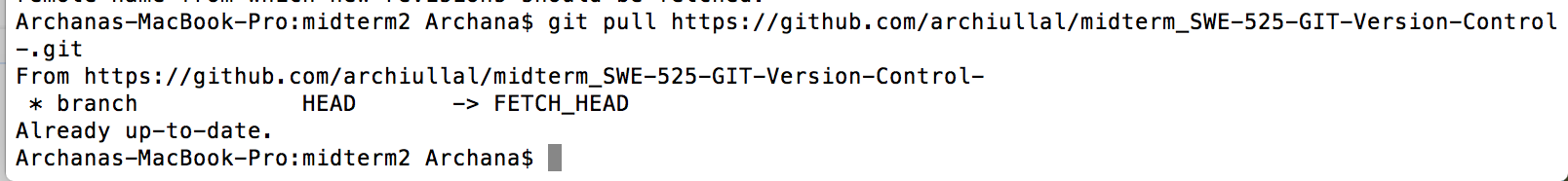


1. Pull the latest changes from the repository to get the updates from others in to your local repo and merge the changes

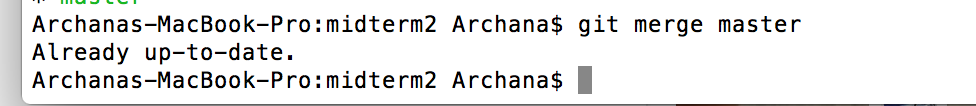


Then I pulled the changes.

**git pull**

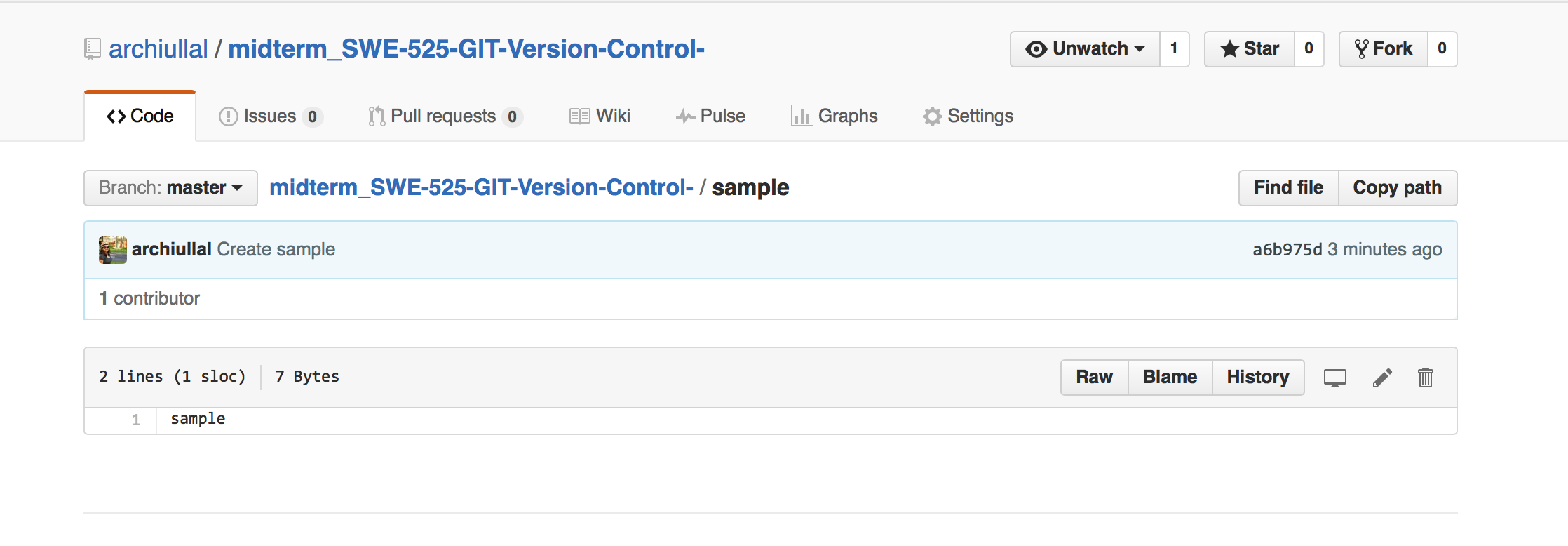


Then merge the changes

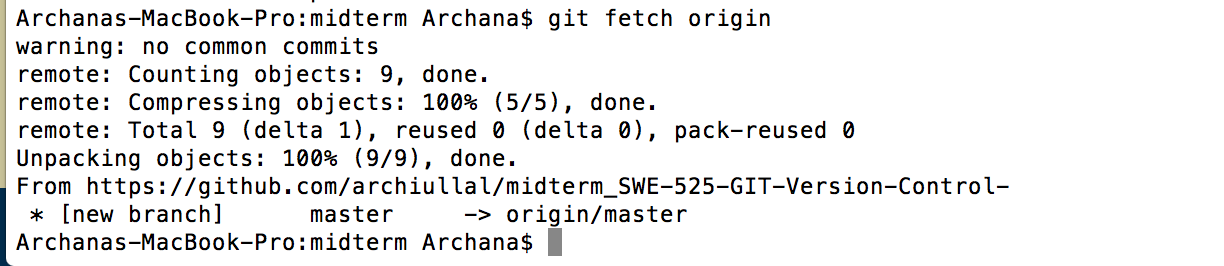


1. Try fetching the changes and perform the merge to get the difference between the pull and the merge command

I have created another file in github.



Then fetch



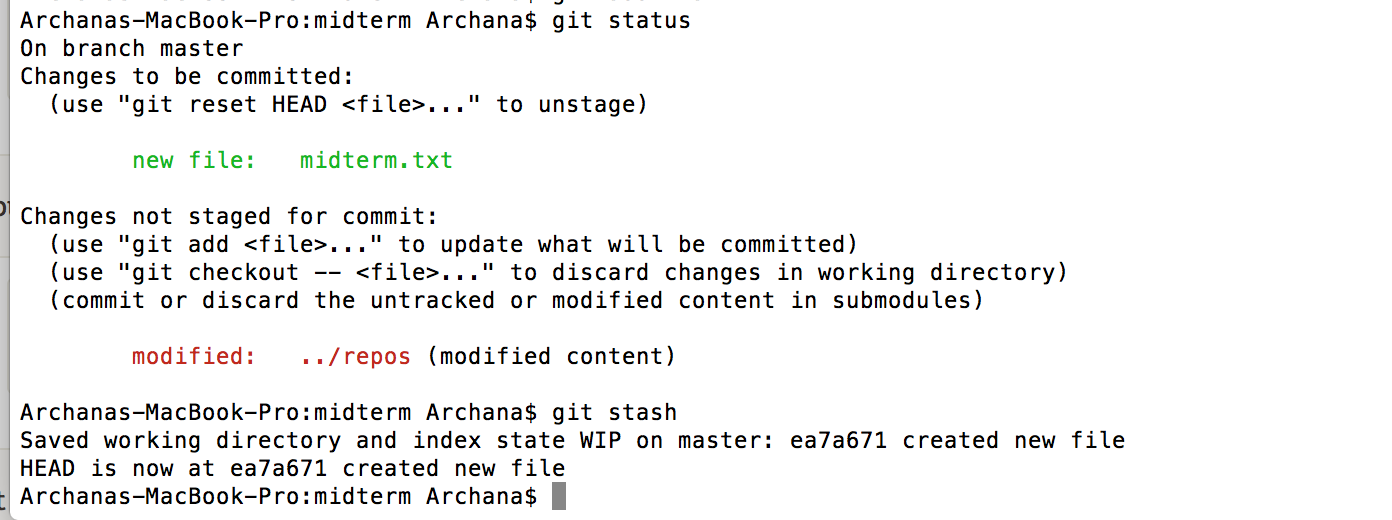
**In the simplest terms, git pull does a git fetch followed by a git merge.**

**SO instead of pulling first and then merging we are using fetch that does both work.**

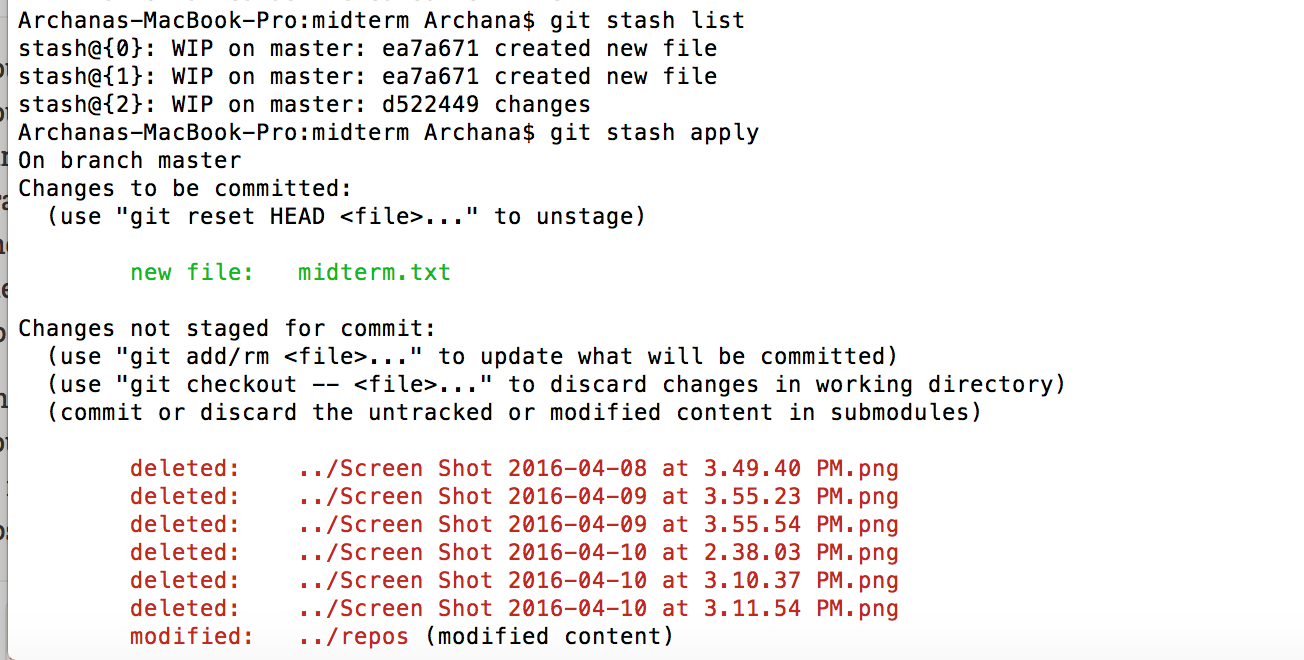
1. Perform some changes and before committing the changes, stash your changes and then pull the changes and finally apply the changes to understand how stashing works

**For this I have created some files.**

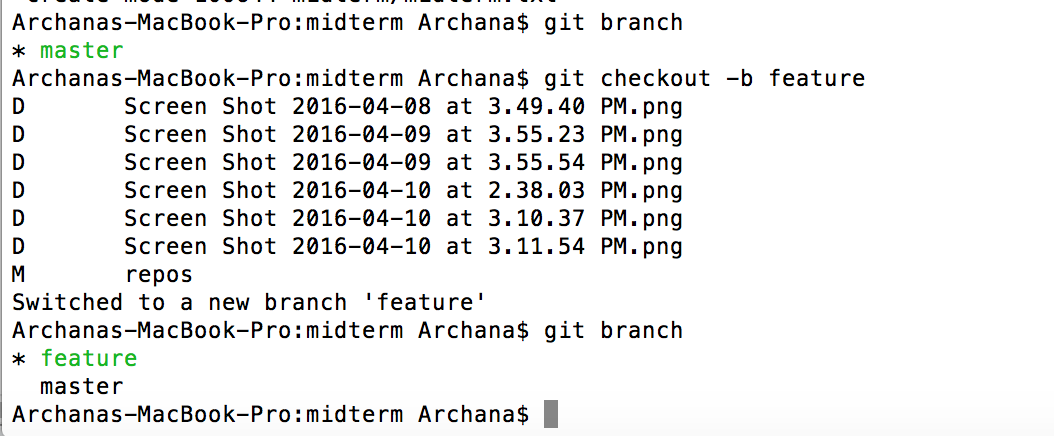
**Then added to the staging area**



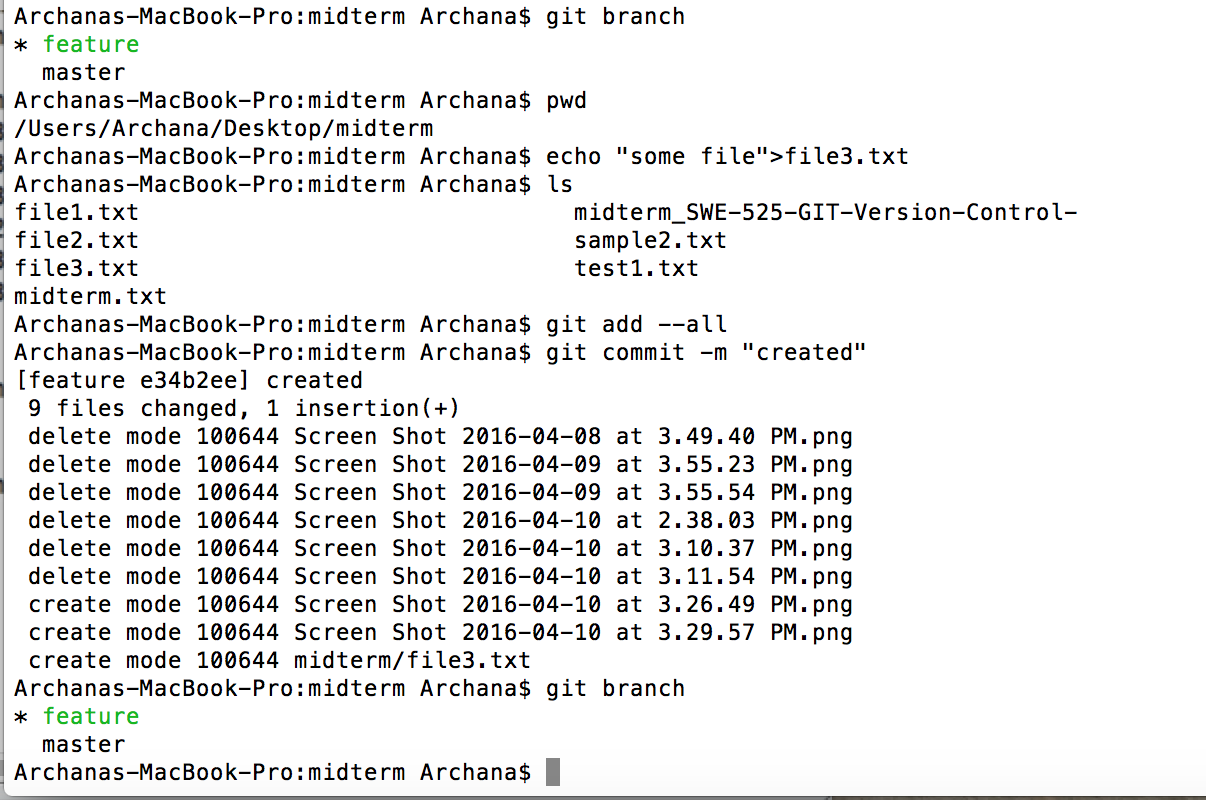
**The finally apply the changes**



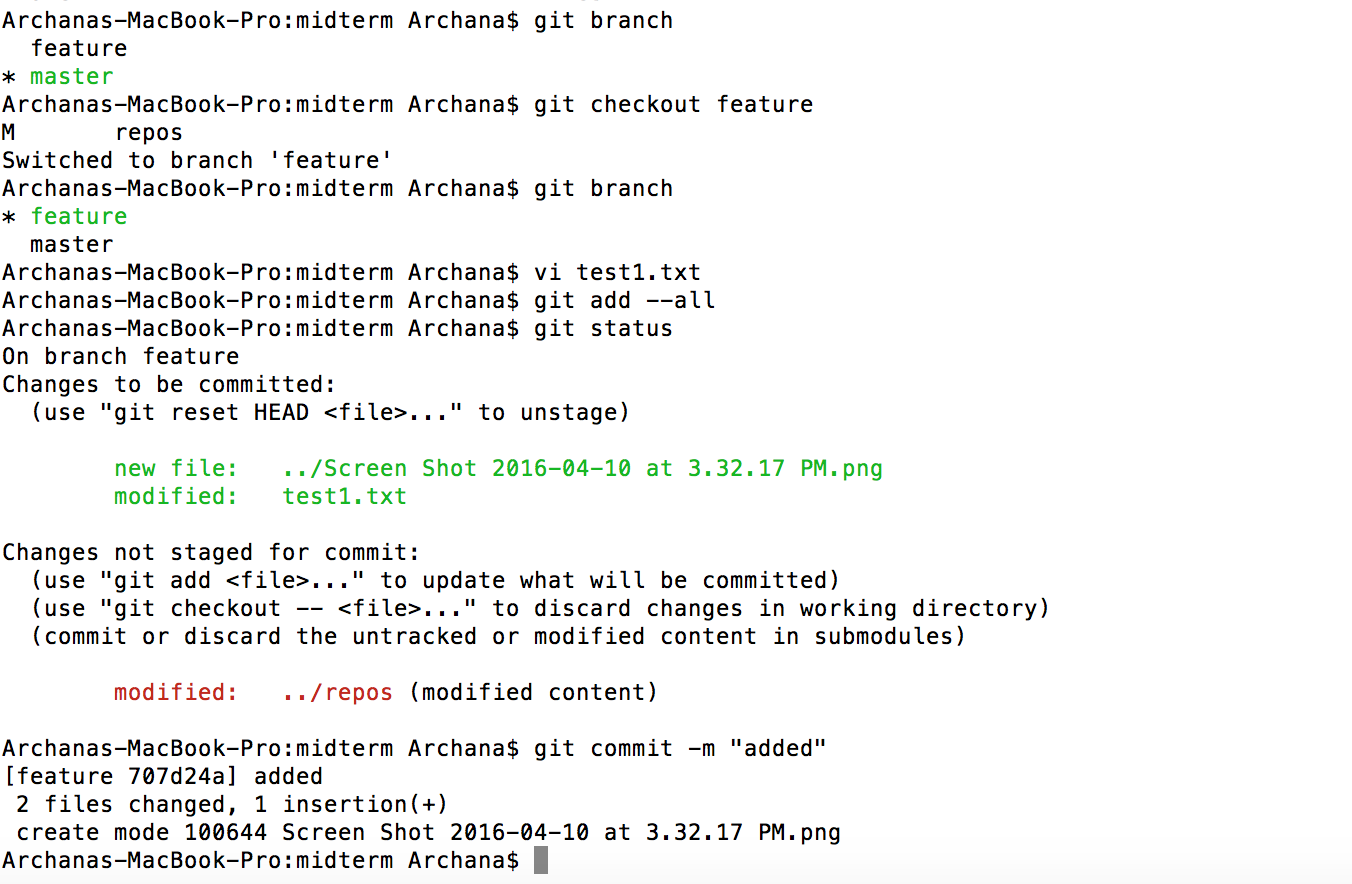
1. Create a feature branch and do some file operations in the branch and commit the changes to the branch



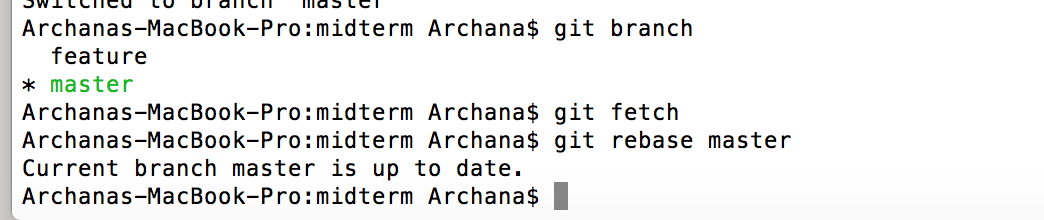
Created some files and commit the changes to the branch



1. Merge the changes using the rebase command and finally perform a safe deletion of the feature branch

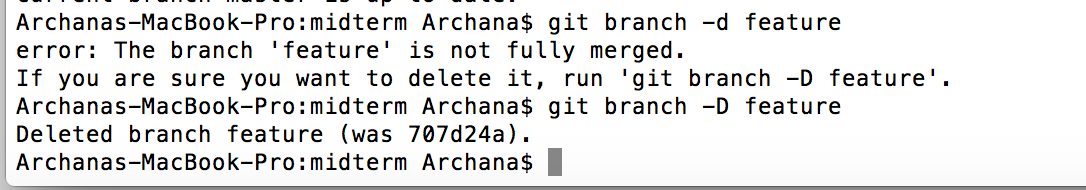


Then git rebase



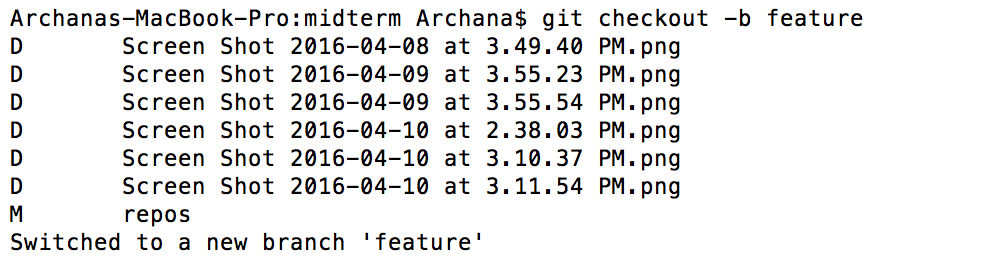
**The major benefit of rebasing is that you get a much cleaner project history.**

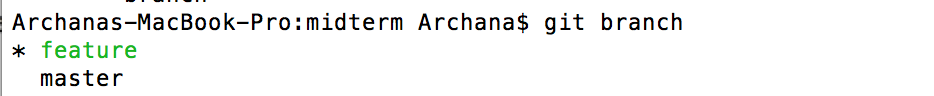
Then safe delete the feature branch



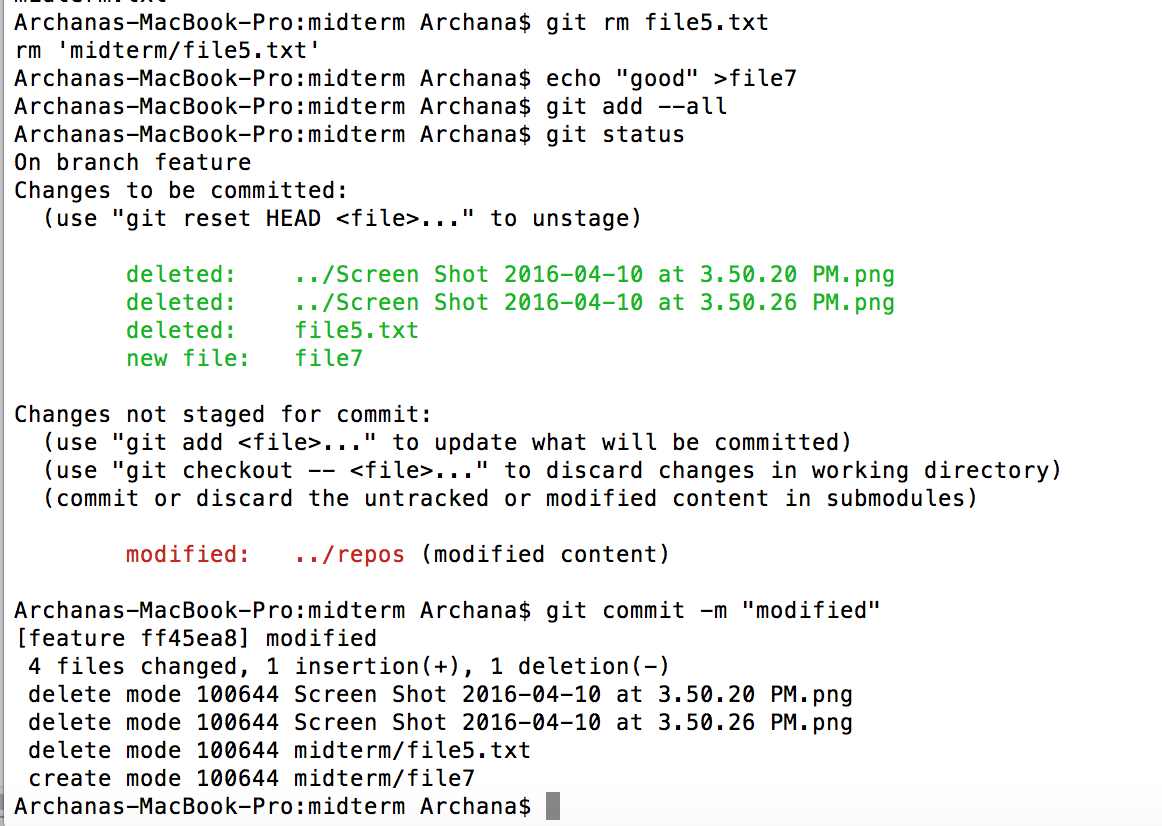
1. Create another feature branch and this time after committing the changes to the feature branch, merge the changes using fast forward merge and then delete the feature branch

**Created another branch called feature.**

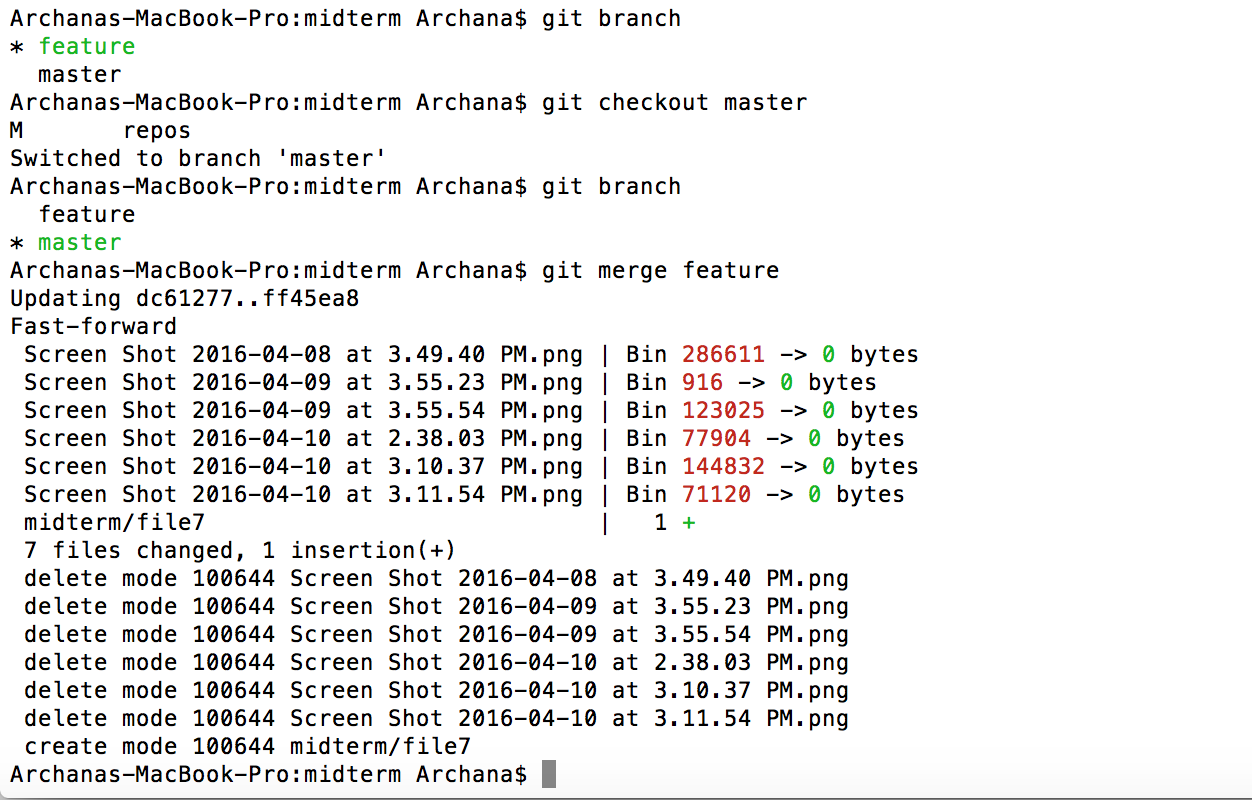




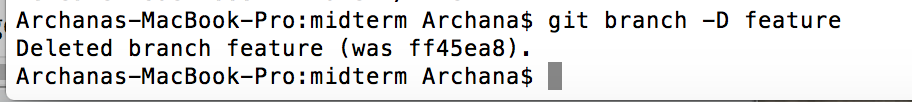
Added some file and modified some files.



**Merge the changes using fast forward merge**



Then delete the branch

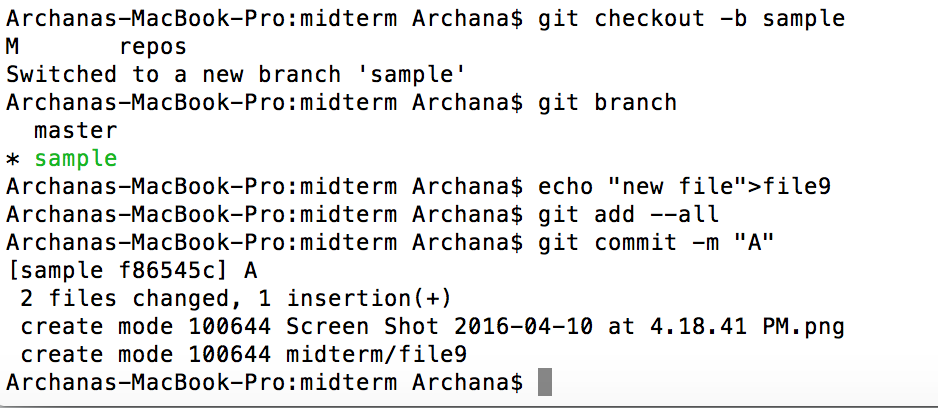


3. Perform following branching and merging operations. Please put commands, screenshots and other information where needed. More detailed oriented response will get higher points. Give me your github repo link also with following steps added.

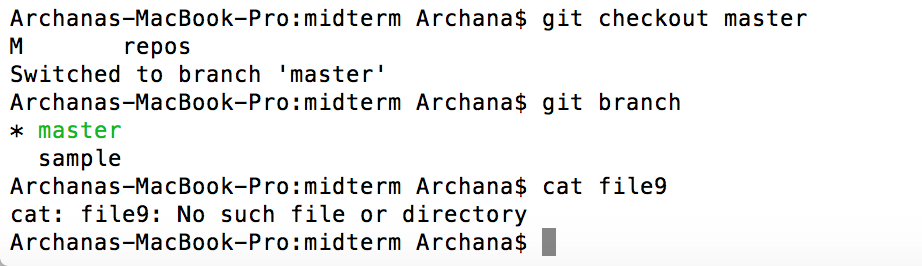
1. Create a local branch using git checkout -b branchname command

**git checkout –b sample**

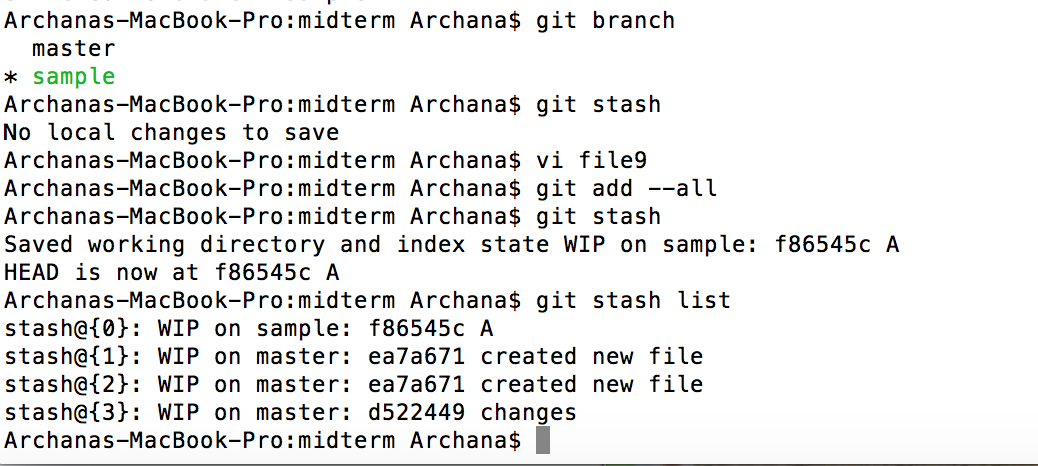
1. Observe the difference by doing some file operations and switch back to the master  branch and see if you can see the changes done on the branch

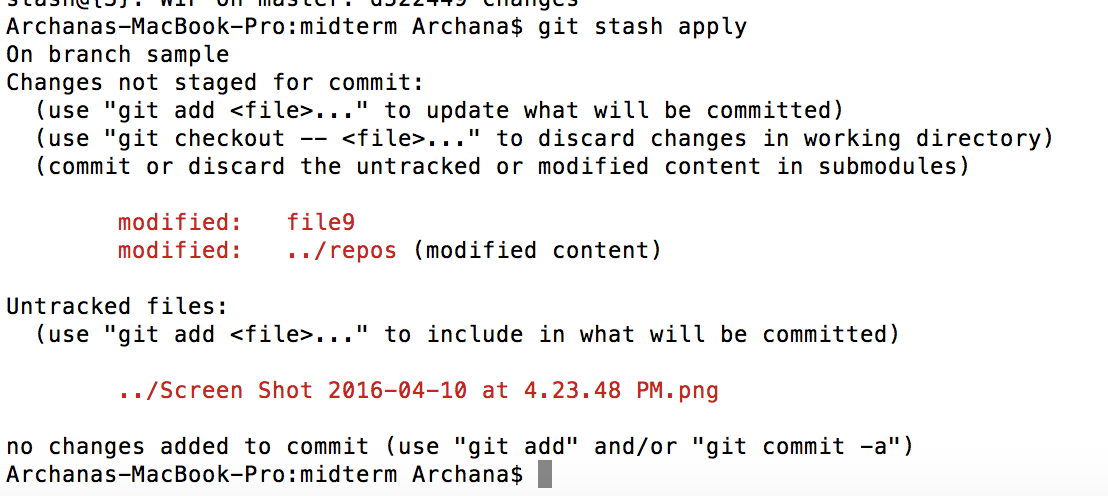


1. Now switch back to the branch and commit the changes and switch to master  branch. Now see if you can still see the changes in the master branch



1. Now switch back to the branch name and stash the changes and apply the changes  to the master branch by switching to the master branch





1. Try merging the changes from the branch to the master branch using all the three  merge strategies and then view the git log

3 merge strategies are:

1. **Pre merge checks**
2. **fast forward merge**
3. **true merge**

**git log**

1. Push the local branch to the remote repository and see if the branch is present on the  remote repository - Github