GHOUSIA INSTITUTE OF TECHNOLOGY FOR WOMEN

NEAR DAIRY CIRCLE, HOSUR ROAD, BENGALURU-560029, KARNATAKA
AFFILIATED TO VTU., BELAGAVI, RECOGNIZED BY GOVERNMENT OF KARNATAKA & A.I.C.T.E., NEW DELHI

Dr. NAVEED
Assistant Professor
GITW-Bengaluru

Project Management with Git-BCS358C

GitHub is a web-based platform used primarily for version control and collaborative software development. It is built around Git, an open-source version control system that tracks changes in files and allows multiple people to work on a project simultaneously without interfering with each other's work.GitHub is widely used by individual developers, teams, and large organizations to collaborate on projects, share code, and contribute to open-source software.

MORE INFORMATION
www.github.com

THIRD SEMESTER
B.E DEGREE 2024

Add **Git Bash**
to
Windows Terminal

GHOUSIA INSTITUTE OF TECHNOLOGY FOR WOMEN

Near Dairy Circle, Hosur Road, Bengaluru , Karnataka 560029

Affiliated to VTU., Belagavi, Recognized by Government of Karnataka & A.I.C.T.E., New Delhi

PROJECT MANAGEMENT WITH GIT

(BCS358C)

As per 2022 Scheme Syllabus Prescribed by V.T.U.

For

THIRD SEMESTER

COMPUTER SCIENCE & ENGINEERING / INFORMATION SCIENCE & ENGINEERING

(Bachelor of Engineering)

Dr.NAVEED M.Tech., PhD.

Assistant Professor

Department of Computer Science & Engineering

GHOUSIA INSTITUTE OF TECHNOLOGY FOR WOMEN

Near Dairy Circle, Hosur Road, Bengaluru-560029, KARNATAKA
Affiliated to VTU., Belagavi, Recognized by Government of Karnataka & A.I.C.T.E., New Delhi

PROJECT MANAGEMENT WITH GIT / BCS358C / THIRD SEMESTER / B.E DEGREE / 2024-25

CERTIFICATE

This is to certify that Miss.______________________________bearing USN____________________of ____________________________ Branch completed the academic requirements for the practical course work titled "**PROJECT MANAGEMENT WITH GIT/ BCS358C**" of THIRD SEMESTER B.E, prescribed by Visvesvaraya Technological University , Belagavi, for the academic year **2024-25**. The details of Mark's obtained by the candidate is given below.

Sl.No	Particulars		Max.Marks (Execution+Record)	Marks Obtained		Page No	Staff Sign
1	Expt-01	Basic Setup and Creation of a New Repository	5+5 = 10			11	
2	Expt-02	To add README.md file into the Repository				15	
3	Expt-03	Creating and Managing Branches				19	
4	Expt-04	Merging of Feature-Branch into the Master Branch				26	
5	Expt-05	Updating of files in the feature-branch				28	
6	Expt-06	Light Weight and Annotated Tags				34	
7	Expt-07	Analyzing GIT History				39	
8	Expt-08	GIT Cherry-pick and Revert				44	
9	Expt-09	Git in VS Code				56	
10	Expt-10	VS Code and Github (cloning of repository)				65	
11	Expt-11	VS Code and Github (creating of new repository)				78	
12	Assignment Experiments		20+20 = 40			86	
Total Marks-A			150				
Test Marks-B			100				
Final Internal Assessment Mark's.			[(A*30)/150] + (B*20%) = **50**				

Internal Assessment Marks Awarded in Words: ________________________________
Signature of Staff Incharge with Date:

Project Management with Git		Semester	3
Course Code	**BCS358C**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0: 0 : 2: 0	SEE Marks	50
Credits	01	Exam Marks	100
Examination type (SEE)	Practical		

Course objectives:

- .To familiar with basic command of Git
- To create and manage branches
- To understand how to collaborate and work with Remote Repositories
- To familiar with virion controlling commands

Sl.NO	Experiments
1	**Setting Up and Basic Commands** Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.
2	**Creating and Managing Branches** Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."
3	**Creating and Managing Branches** Write the commands to stash your changes, switch branches, and then apply the stashed changes.
4	**Collaboration and Remote Repositories** Clone a remote Git repository to your local machine.
5	**Collaboration and Remote Repositories** Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.
6	**Collaboration and Remote Repositories** Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.
7	**Git Tags and Releases** Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.
8	**Advanced Git Operations**

	Write the command to cherry-pick a range of commits from "source-branch" to the current branch.
9	**Analysing and Changing Git History** Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?
10	**Analysing and Changing Git History** Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."
11	**Analysing and Changing Git History** Write the command to display the last five commits in the repository's history.
12	**Analysing and Changing Git History** Write the command to undo the changes introduced by the commit with the ID "abc123".

Course outcomes (Course Skill Set):
At the end of the course the student will be able to:

- Use the basics commands related to git repository
- Create and manage the branches
- Apply commands related to Collaboration and Remote Repositories
- Use the commands related to Git Tags, Releases and advanced git operations
- Analyse and change the git history

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.

- Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled down to **30 marks** (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
- In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The marks scored shall be scaled down to **20 marks** (40% of the maximum marks).

The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

- SEE marks for the practical course are 50 Marks.
- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.
- The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.

- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.
- General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)
- Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.

The minimum duration of SEE is 02 hours

Suggested Learning Resources:

- Version Control with Git, 3rd Edition, by Prem Kumar Ponuthorai, Jon Loeliger Released October 2022, Publisher(s): O'Reilly Media, Inc.
- Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, https://git-scm.com/book/en/v2
- https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_0130944433473699842782_shared/overview
- https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_01330134712177459211926_shared/overview

INRODUCTION TO GIT

Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It was created by Linus Torvalds in 2005 for the development of the Linux kernel.

AboutVersionControl

Whatis“versioncontrol”,andwhyshouldyoucare?Versioncontrolisa system that recordschangestoafileorsetoffilesovertimesothatyoucanrecallspecificversionslate r.Fortheexamplesinthisbook,youwillusesoftwaresourcecodeasthefiles being version controlled,thoughinrealityyoucandothiswithnearlyanytypeoffileonacomputer.

Ifyouareagraphicorwebdesignerandwanttokeepeveryversionofanimageorlayout(whichyou would most certainly want to), a Version Control System (VCS) is a very wise thing to use. Itallowsyoutorevertselectedfilesbacktoapreviousstate,reverttheentireprojectback to apreviousstate,comparechangesovertime,seewholastmodifiedsomethingthatmigh tbecausingaproblem,whointroducedanissueandwhen,andmore.UsingaVCSalsog enerallymeansthatifyou screw things up or lose files, you can easily recover. In addition, you get all this for very littleoverhead.

LocalVersionControlSystems

Many people’s version-control method of choice is to copy files into another directory (perhaps atime-stamped directory, if they’re clever). This approach is very common because it is so simple, butitisalsoincrediblyerrorprone.Itiseasytoforgetwhichdirectoryyou’reinandaccid entallywritetothewrongfileorcopyoverfilesyoudon’tmeanto.

To deal with this issue, programmers long ago developed local VCSs that had a simple database thatkeptallthechangestofilesunderrevisioncontrol.

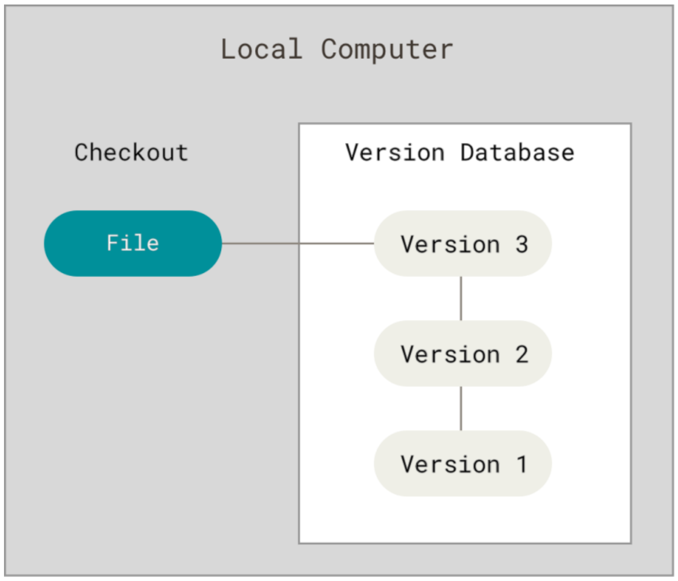


Figure1.Localversioncontroldiagram

One of the most popular VCS tools was a system called RCS, which is still distributed with manycomputers today. RCS works by keeping patch sets (that is, the differences between files) in a specialformatondisk;itcanthenre-createwhatanyfilelookedlikeatanypointintimebyaddingupallthepatches.

CentralizedVersionControlSystems

Thenextmajorissuethatpeopleencounteristhattheyneedtocollaboratewithdevelopersonothersystems.Todealwiththisproblem,CentralizedVersionControlSystems(CVCSs)weredeveloped. These systems (such as CVS, Subversion, and Perforce) have a single server that containsall the versioned files, and a number of clients that check out files from that central place. For manyyears,thishasbeenthestandardforversioncontrol.

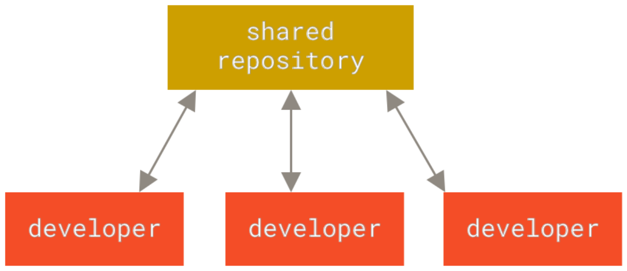


Figure2.Centralizedversioncontroldiagram

This setup offers many advantages, especially over local VCSs. For example, everyone knows to acertain degree what everyone else on the project is doing. Administrators have fine-grained controlover who can do what, and it's far easier to administer a CVCS than it is to deal with local databasesoneveryclient.

However, this setup also has some serious downsides. The most obvious is the single point of failurethat the centralized server represents. If that server goes down for an hour, then during that hournobody can collaborate at all or save versioned changes to anything they're working on. If the harddisk the central database is on becomes corrupted, and proper backups haven't been kept, you loseabsolutely everything — the entire history of the project except whatever single snapshots peoplehappens to have on their local machines. Local VCSs suffer from this same problem — whenever youhavetheentirehistoryoftheprojectinasingleplace,yourisklosingeverything.

DistributedVersionControlSystems

This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as Git, Mercurialor Darcs), clients don't just check out the latest snapshot of the files; rather, they fully mirror therepository,includingitsfullhistory.Thus,ifanyserverdies,andthesesystemswere collaboratingvia that server, any of the client repositories can be copied back up to the server to restore it. Everycloneisreallyafullbackupofallthedata.

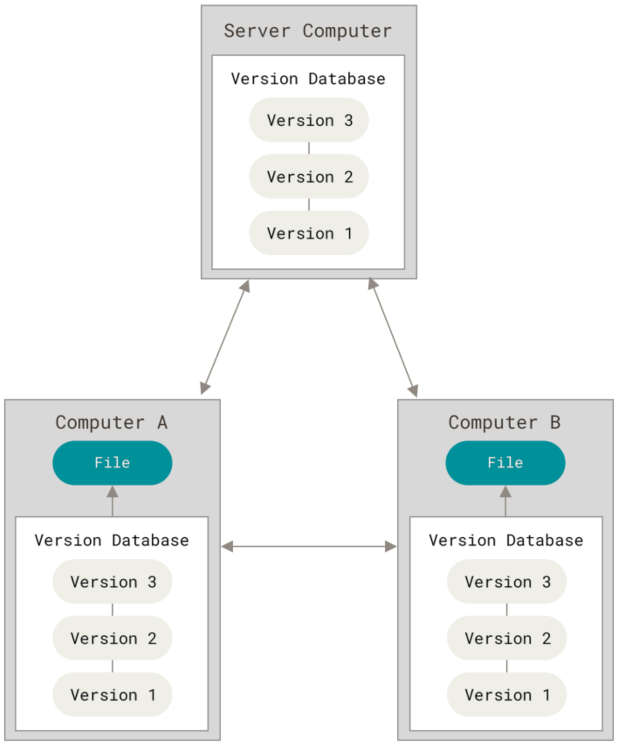


Figure3.Distributedversioncontroldiagram

Furthermore, many of these systems deal pretty well with having several remote repositories theycanworkwith,soyoucancollaboratewithdifferentgroupsofpeopleindifferentw ayssimultaneously within the same project. This allows you to set up several types of workflows thataren'tpossibleincentralizedsystems,suchashierarchicalmodels.

AShortHistoryofGit

Aswithmanygreatthingsinlife,Gitbeganwithabitofcreativedestructionandfierycontroversy.

The Linux kernel is an open-source software project of fairly large scope. During the early years ofthe Linux kernel maintenance (1991–2002), changes to the software were passed around as patchesandarchivedfiles.In2002,theLinuxkernelprojectbeganusingaproprietary DVCS calledBit Keeper.

In2005,therelationshipbetweenthecommunitythatdevelopedtheLinuxkernelandth ecommercialcompanythatdevelopedBit Keeperbrokedown,andthetool’sfree-of-chargestatuswasrevoked.ThispromptedtheLinuxdevelopmentcommunity(andinp articularLinusTorvalds,thecreatorofLinux)todeveloptheirowntoolbasedonsomeo fthelessonstheylearnedwhileusingBit Keeper.Someofthegoalsofthenewsystemwereasfollows:

- Speed
- Simpledesign
- Strongsupportfornon-lineardevelopment(thousandsofparallelbranches)
- Fullydistributed
- AbletohandlelargeprojectsliketheLinuxkernelefficiently(speedanddatasize)

Since its birth in 2005, Git has evolved and matured to be easy to use and yet retain these initialqualities.It’samazinglyfast,it’sveryefficientwithlargeprojects, and it has an incrediblebranchingsystemfornon-lineardevelopment(seeGitBranching).

WhatisGit?

So, what is Git in a nutshell? This is an important section to absorb, because if you understand whatGit is and the fundamentals of how it works, then using Git effectively will probably be much easierforyou.AsyoulearnGit,trytoclearyourmindofthethingsyoumayknowaboutot herVCSs,suchas CVS, Subversion or Perforce — doing so will help you avoid subtle confusion when using the tool.Even though Git’s user interface is fairly similar to these other VCSs, Git stores and thinks aboutinformationinaverydifferentway,andunderstandingthesedifferenceswillhel pyouavoidbecomingconfusedwhileusingit.

Snapshots,NotDifferences
ThemajordifferencebetweenGitandanyotherVCS(Subversionandfriendsincluded)isthewa yGit thinks about its data. Conceptually, most other systems store information as a list of file-basedchanges. These other systems (CVS, Subversion, Perforce, and so on) think

of the information theystore as a set of files and the changes made to each file over time (this is commonly described as*delta-based*versioncontrol).

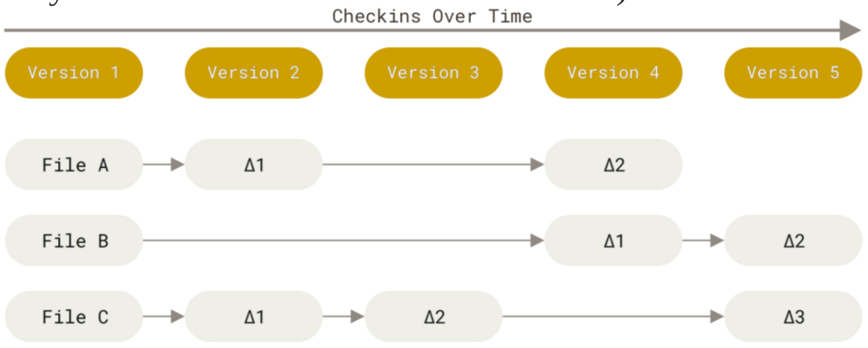
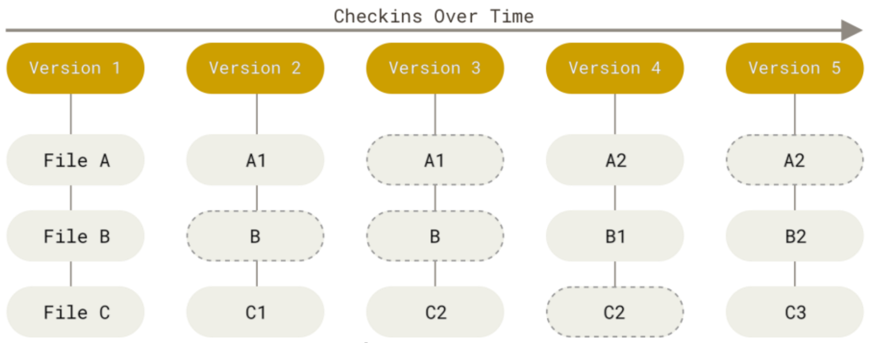


Figure4.Storingdataaschangestoabaseversionofeachfile

Git doesn't think of or store its data this way. Instead, Git thinks of its data more like a series ofsnapshots of a miniature filesystem. With Git, every time you commit, or save the state of yourproject, Git basically takes a picture of what all your files look like at that moment and stores areference to that snapshot. To be efficient, if files have not changed, Git doesn't store the file again,just a link to the previous identical file it has already stored. Git thinks



about its data more like a**streamofsnapshots**.

Figure5.Storingdataassnapshotsoftheprojectovertime

ThisisanimportantdistinctionbetweenGitandnearlyallotherVCSs.ItmakesGitreconsideralmosteveryaspectofversioncontrolthatmostothersystemscopied from the previousgeneration. This makes Git more like a mini filesystem with some incredibly powerful tools built ontopofit,ratherthansimplyaVCS.We'llexploresomeofthebenefitsyougainbythinkingofyourdatathiswaywhenwecoverGitbranchinginGitBranching.

NearlyEveryOperationIsLocal

Most operations in Git need only local files and resources to operate — generally no information isneeded from another computer on your network. If you're used to a CVCS where most operationshavethatnetworklatencyoverhead,thisaspectofGitwillmakeyouthinkthatthegodsofspeedhave blessed Git with unworldly powers. Because you have the entire history of the project rightthereonyourlocaldisk,mostoperationsseemalmostinstantaneous.

Forexample,tobrowsethehistoryoftheproject,Gitdoesn’tneedtogoouttotheserv ertogetthehistory and display it for you — it simply reads it directly from your local database. This means yousee the project history almost instantly. If you want to see the changes introduced between thecurrentversionofafileandthefileamonthago,Gitcanlookupthefileamonthago anddoalocaldifference calculation, instead of having to either ask a remote server to do it or pull an olderversionofthefilefromtheremoteservertodoitlocally.

This also means that there is very little you can’t do if you’re offline or off VPN. If you get on anairplaneoratrainandwanttodoalittlework,youcancommithappily(toyour*local* copy,remember?) until you get to a network connection to upload. If you go home and can’t get your VPNclient working properly, you can still work. In many other systems, doing so is either impossible orpainful. In Perforce, for example, you can’t do much when you aren’t connected to the server; inSubversionandCVS,youcaneditfiles,butyoucan’tcommitchangestoyourdatabas e(becauseyour database is offline). This may not seem like a huge deal, but you may be surprised what a bigdifferenceitcanmake.

GitHasIntegrity

Everything in Git is check summed before it is stored and is then referred to by that checksum. Thismeans it’s impossible to change the contents of any file or directory without Git knowing about it.This functionality is built into Git at the lowest levels and is integral to its philosophy. You can’t loseinformationintransitorgetfilecorruptionwithoutGitbeingabletodetectit.

mechanism that Git uses for this checksumming is called a SHA-1 hash. This is a 40-characterstring composed of hexadecimal characters (0–9 and a–f) and calculated based on the contents of afileordirectorystructureinGit.ASHA-1hashlookssomethinglikethis:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Youwill seethesehash valuesall overtheplace in Git because it uses themsomuch. Infact, Gitstoreseverythinginitsdatabasenotbyfilenamebutbythehashvalueofitscontents.

Git GenerallyOnlyAdds Data

When you do actions in Git, nearly all of them only *add* data to the Git database. It is hard to get thesystem to do anything that is not undoable or to make it erase data in any way. As with any VCS, youcanloseormessupchangesyouhaven’tcommittedyet,butafteryoucommitasnapshotintoGit,itisverydifficulttolose,especiallyifyouregularlypushyourdatabasetoanotherrepository.

This makes using Git a joy because we know we can experiment without the danger of severelyscrewingthingsup.Foramorein-depthlookathowGitstoresitsdataandhowyoucanrecoverdatathatseemslost,seeUndoingThings.

TheThreeStates

Pay attention now — here is the main thing to remember about Git if you want the rest of yourlearning process to go smoothly. Git has three main states that your files can reside in: *modified*,*staged*,and*committed*:

- Modifiedmeansthatyouhavechangedthefilebuthavenotcommittedittoyourdatabaseyet.
- Stagedmeansthatyouhavemarkedamodifiedfileinitscurrentversiontogointoyournextcommitsnapshot.
- Committedmeansthatthedataissafelystoredinyourlocaldatabase.

ThisleadsustothethreemainsectionsofaGitproject:theworkingtree,thestagingarea, andtheGitdirectory.

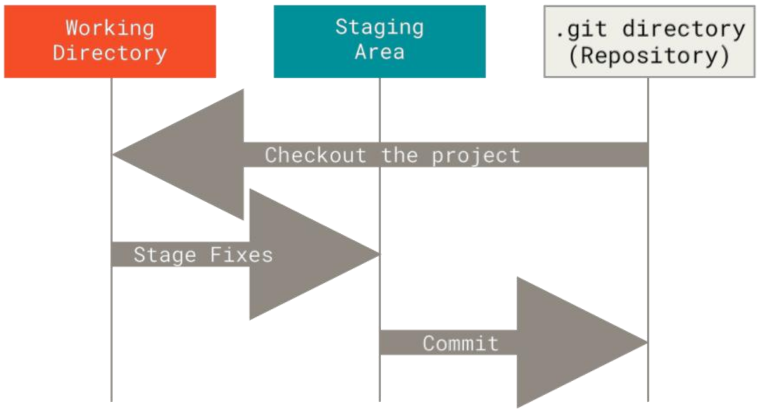


Figure6.Workingtree, staging area, andGitdirectory

The working tree is a single checkout of one version of the project. These files are pulled out of thecompresseddatabaseintheGitdirectoryandplacedondiskforyoutouseormodify.

Thestagingareaisafile,generallycontainedinyourGitdirectory,thatstoresinformationaboutwhat will go into your next commit. Its technical name in Git parlance is the “index”, but the phrase“stagingarea”worksjustaswell.

The Git directory is where Git stores the metadata and object database for your project. This is themost important part of Git, and it is what is copied when you*clone* a repository from anothercomputer.

ThebasicGitworkflowgoessomethinglikethis:

1. Youmodifyfilesinyourworkingtree.
2. Youselectivelystagejustthosechangesyouwanttobepartofyournextcommit,whichadds *only*thosechangestothestagingarea.
3. Youdoacommit,whichtakesthefilesastheyareinthestagingareaandstoresthatsn apshotpermanentlytoyourGitdirectory.

IfaparticularversionofafileisintheGitdirectory,it’sconsidered*committed*.Ifithasbe enmodified and was added to the staging area, it is *staged*. And if it was changed since it was checkedoutbuthasnotbeenstaged,itis*modified*.InGitBasics,you’lllearnmoreaboutt hesestatesandhowyoucaneithertakeadvantageofthemorskipthestagedpartentirely.

Uses of Git

1. Collaboration.
Teams: Multiple developers can work on the same project simultaneously without conflicts.
Open Source: Git is popular in the open-source community, allowing developers from around the world to contribute to a project.

2. Tracking Changes
History: Git maintains a history of changes, which can be reviewed and reverted if necessary.
Blame: Identify who made specific changes to the code and when they were made.

3. Branching and Merging
Feature Development: Developers can create separate branches for new features, bug fixes, or experiments without affecting the main codebase.
Code Reviews: Changes can be reviewed in branches before merging them into the main branch.

4. Backup and Restore
Local Repositories: Each user has a complete copy of the repository, providing a backup of the project.
Remote Repositories: Remote repositories (like those hosted on GitHub, GitLab, or Bitbucket) provide additional backups and a centralized place to push changes.

5. Continuous Integration/Continuous Deployment (CI/CD)
Automation: Git can be integrated with CI/CD pipelines to automate testing, building, and deployment processes whenever changes are pushed to the repository.

6. Documentation

README Files: Git repositories often include README files that provide information about the project, how to set it up, and how to contribute.

Wikis: Many Git hosting services provide integrated wikis for detailed documentation.

Git is a powerful tool that has become essential for modern software development, enabling efficient collaboration, robust change tracking, and streamlined workflows.

InstallingGit

Before you start using Git, you have to make it available on your computer. Even if it's alreadyinstalled,it'sprobablyagoodideatoupdatetothelatestversion.Youcaneitherinstallitasapackageorviaanotherinstaller,ordownloadthesourcecodeandcompileityourself.

InstallingonWindows

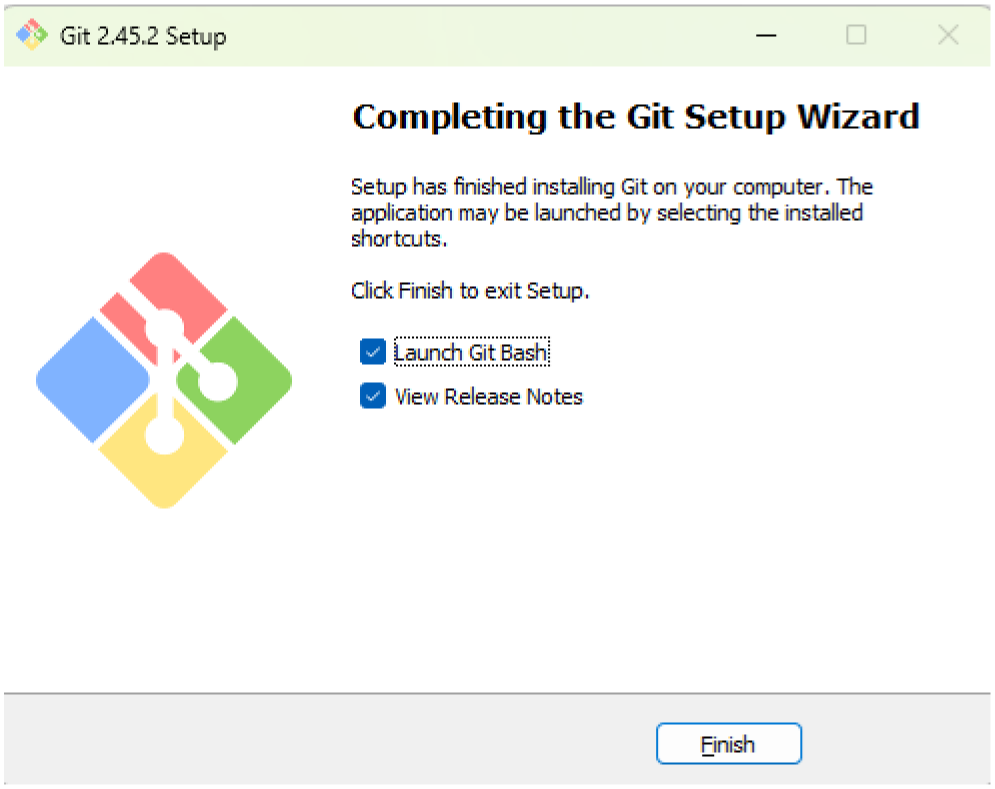
Therearealsoafew ways to install Git on Windows.The most official build is available fordownloadontheGitwebsite.Justgo tohttps://git-scm.com/download/winand the download willstartautomatically.Notethatthisis a project called Git for Windows, which is separate from Gititself;formoreinformationonit,gotohttps://gitforwindows.org.

To get an automated installation you can use the Git Chocolatey package. Note that the Chocolateypackageiscommunitymaintained.

OR to Download click below link

https://drive.google.com/file/d/1H9ZMW2lZnqNngLv-PTXSbUUXPas56IVw/view?usp=sharing

Go through the installation process by clicking Next for all the steps at the last click on Launch GitBash


Git 2.45.2 Setup

Completing the Git Setup Wizard

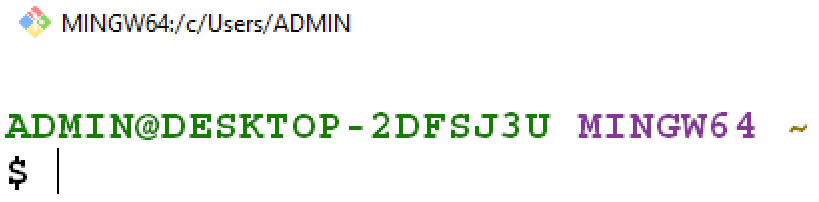
Setup has finished installing Git on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

[✓] Launch Git Bash

[✓] View Release Notes

Click on Finish.



MINGW64:/c/Users/ADMIN

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ 
```

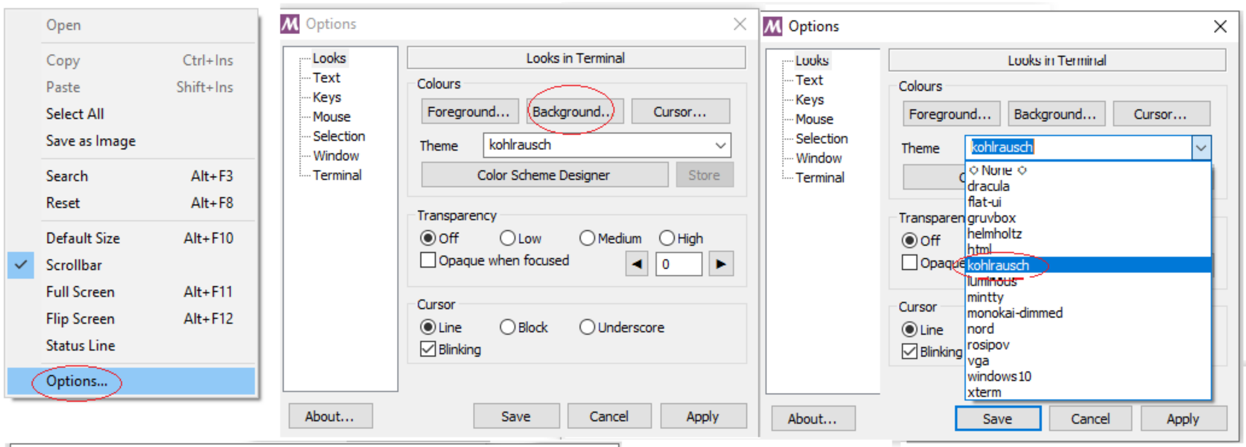
EXPERIMENT-01 Basic Setup and Creation of a New Repository

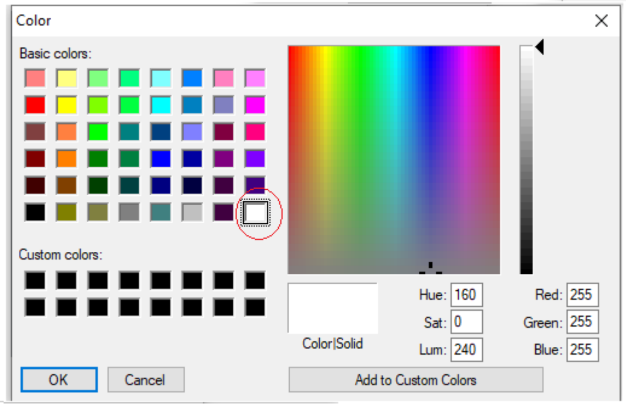
Aim:

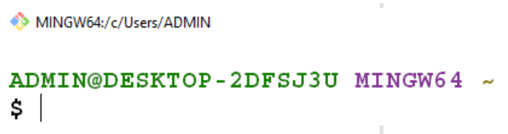
1. To create a new repository “1WT23CS000” under any disc drive such as Z drive and also a sub repository “aboutMyself”.
2. To make the default branch as master branch in some systems it is main branch.
3. To launch the git and enter the user configurations like name, email ID.

First-TimeGitSetup

To make the background color white and to make the theme Kohlrausch.







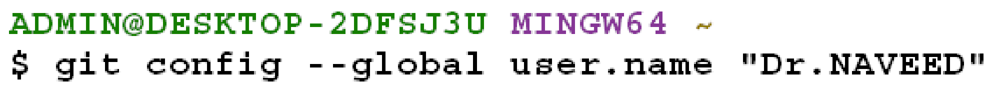
```
MINGW64:/c/Users/ADMIN

ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ |
```

NowthatyouhaveGitonyoursystem,you’llwanttodoafewthingstocustomizeyourGitenvironment. You should have to do these things only once on any given computer; they’ll stickaroundbetweenupgrades.Youcanalsochangethematanytime by running through thecommandsagain.
YourIdentity
ThefirstthingyoushoulddowhenyouinstallGitistosetyourusernameandemailaddress.Thisis important because every Git commit uses this information, and it’s immutably baked into thecommitsyoustartcreating:
Again,youneedtodothisonlyonceifyoupassthe--globaloption,becausethenGitwillalwaysuse that information for anything you do on

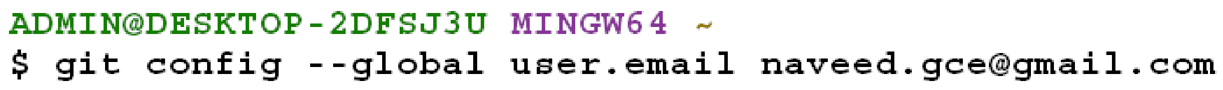
```
$ git config --global user.name "Dr.NAVEED"
$ git config --global user.email naveed.gce@gmail.com
```

that system. If you want to override this with adifferentnameoremailaddressforspecificprojects,youcanrunthecommandwithoutthe-

-globaloptionwhenyou’reinthatproject.

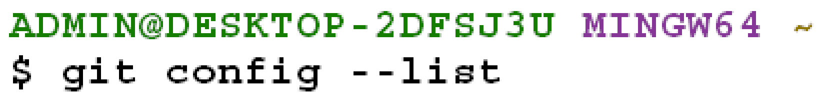


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ git config --global user.name "Dr.NAVEED"
```



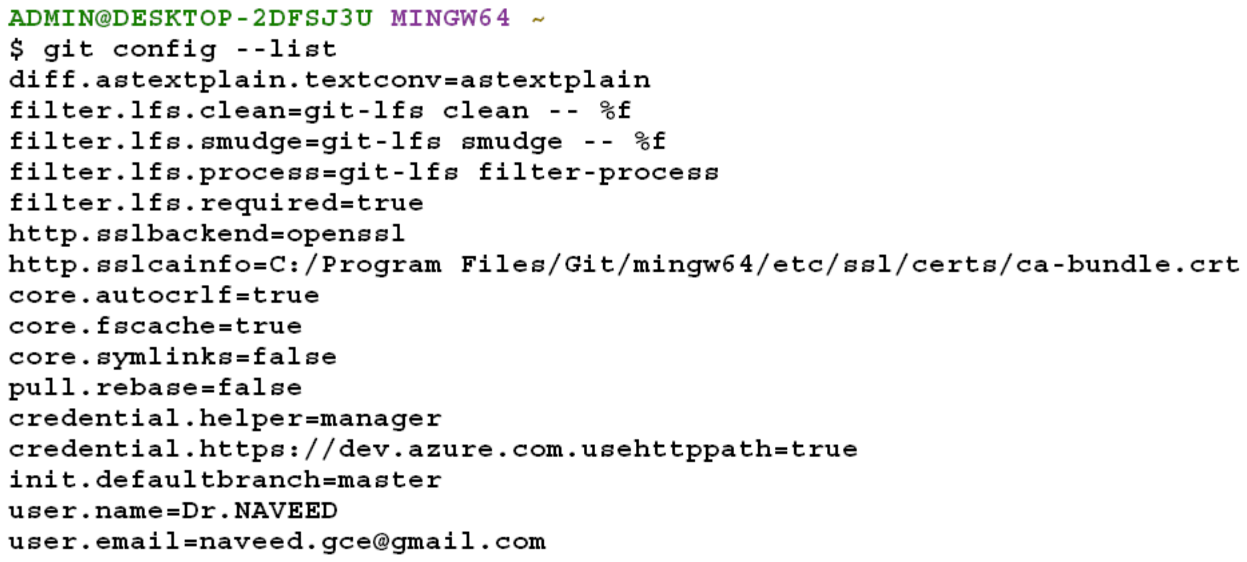
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ git config --global user.email naveed.gce@gmail.com
```

CheckingYourSettings
Ifyouwanttocheckyourconfigurationsettings,youcanusethegitconfig--listcommandtolistallthesettingsGitcanfindatthatpoint:



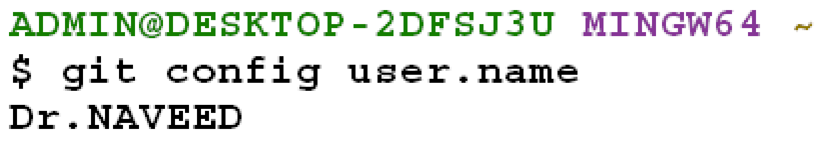
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ git config --list
```

It will show:



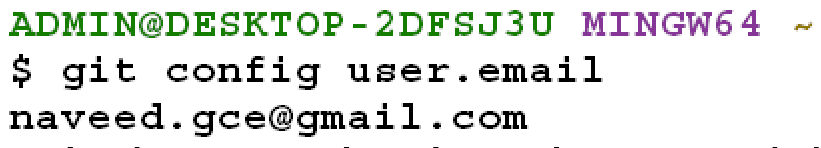
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=Dr.NAVEED
user.email=naveed.gce@gmail.com
```

It will show all details and at the end user name and email ID will be highlighted. If only user nameis required then git config user.name will be used.



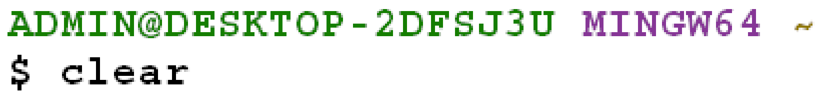
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ git config user.name
Dr.NAVEED
```

To check email of the user use git config user.email



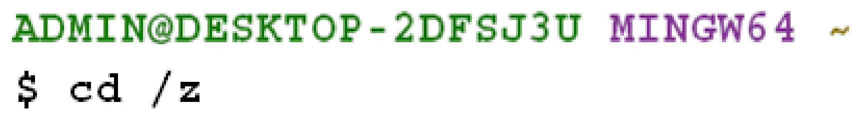
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ git config user.email
naveed.gce@gmail.com
```

To clear the screen, use clear. Alternatively, you can use the keyboard shortcut Ctrl + L which also clears the screen in most terminal emulators, including Git Bash.



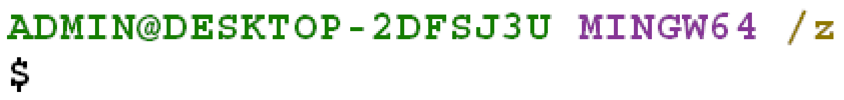
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ clear
```

To Create a New Repository (Folder): By name “1WT23CS000”.
First specify in which drive of your computer you want to create the new repository for Example ‘Z’ Drive



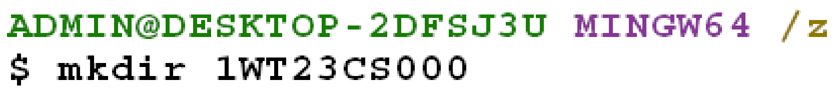
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ cd /z
```

In the next line you can see that Z drive is highlighted.



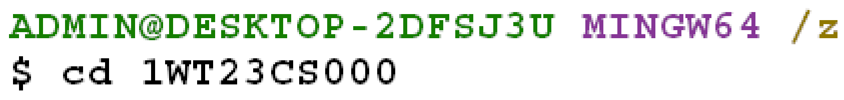
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z
$
```

To create a new repository by name 1WT23CS000



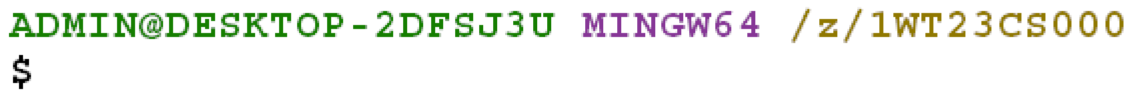
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z
$ mkdir 1WT23CS000
```

To switch to the new repository



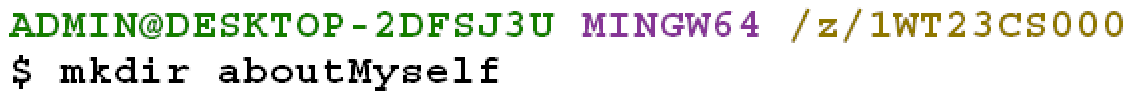
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z
$ cd 1WT23CS000
```

You can see now



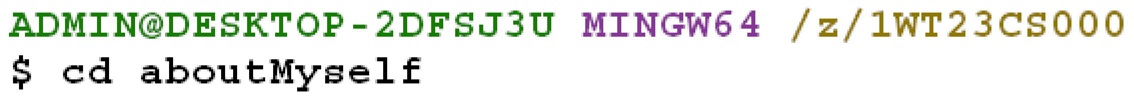
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000
$
```

To create a sub folder “ aboutMyself” inside the main folder



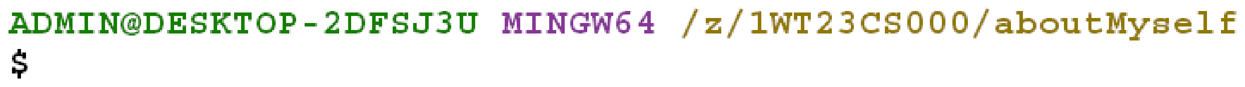
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000
$ mkdir aboutMyself
```

To shift to the subfolder



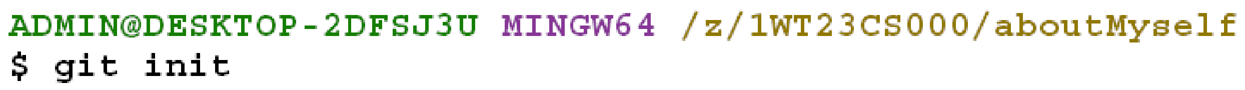
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000
$ cd aboutMyself
```

Now you can see



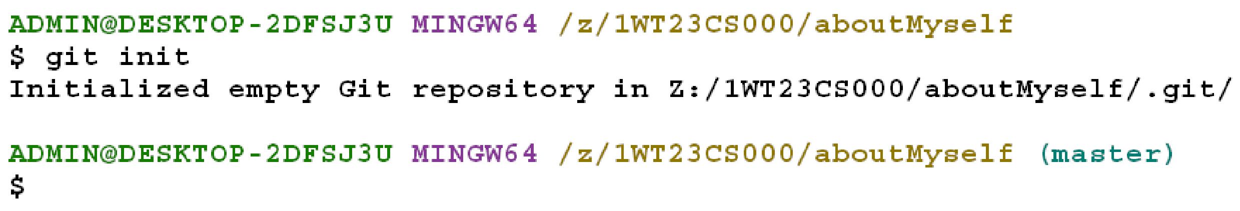
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself
$
```

Once the repository is created initialize the git:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself
$ git init
```

It will make master branch as default branch.



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself
$ git init
Initialized empty Git repository in Z:/1WT23CS000/aboutMyself/.git/

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$
```

OUTPUT:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git config user.name
Dr.NAVEED

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git config user.email
naveed.gce@gmail.com
```

EXPERIMENT No: 02 To add README.md file into the Repository

Aim:

1. To add README.md file into the repository /z/1WT23CS000/aboutMyself under master branch.
2. To add the contents given below:
```
Title: Dr.
Full Name: Naveed
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with GIT
Subject Code: BCS358C
Academic Year: 2024-25
Mobile No: 9620483405
Email ID: naveed.gce@gmail.com
```
3. To commit with a message "Contents updated successfully"

First get back to the repository by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ cd /z/1WT23CS000/aboutMyself
```

Now the repository is ready:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$
```

To Add README.md file inside the sub folder

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git add README.md
```

To check the file added:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git ls-files
```

It will show:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git ls-files
README.md
```

Note:Some times if it shows some error like Z drive is unsafe directory then to make it safe use:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git config --global --add safe.directory z:/
```

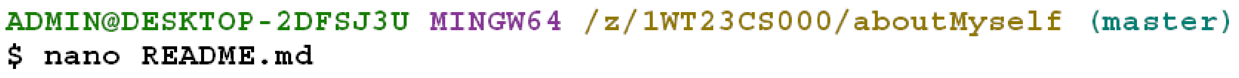
If git add command shows some error, then use:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ echo > README.md
```

This command creates a file named README.md but it will be untraced. To make it traced use

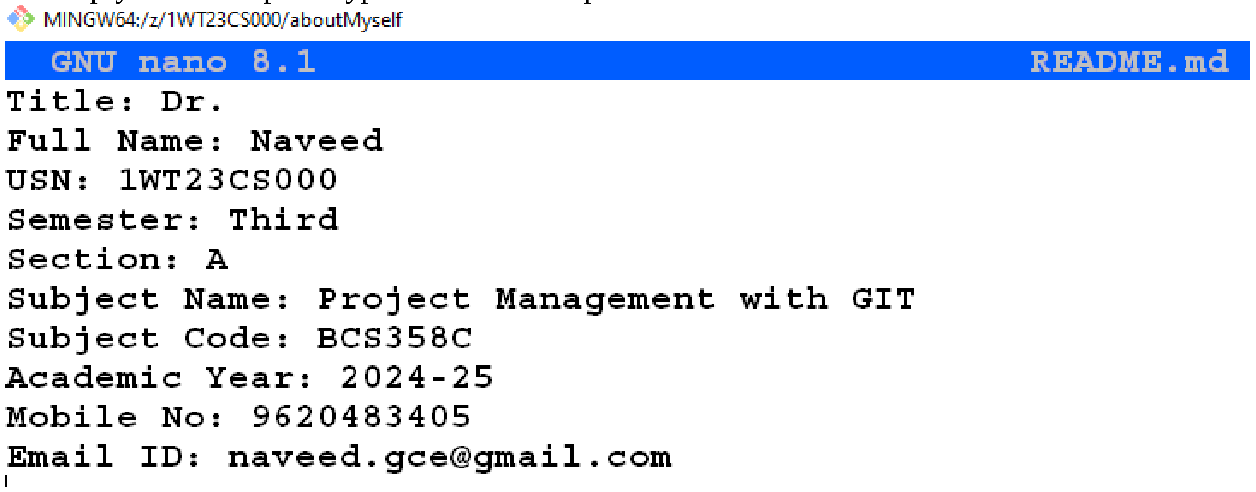
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git add README.md
```

To open the README.md file.



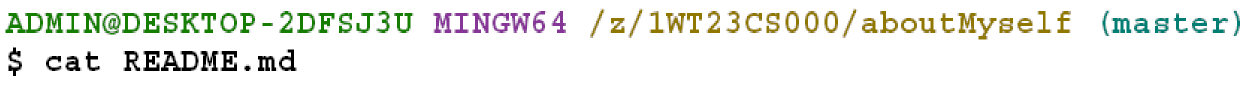
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ nano README.md
```

An empty file will open. Type the below sample data:



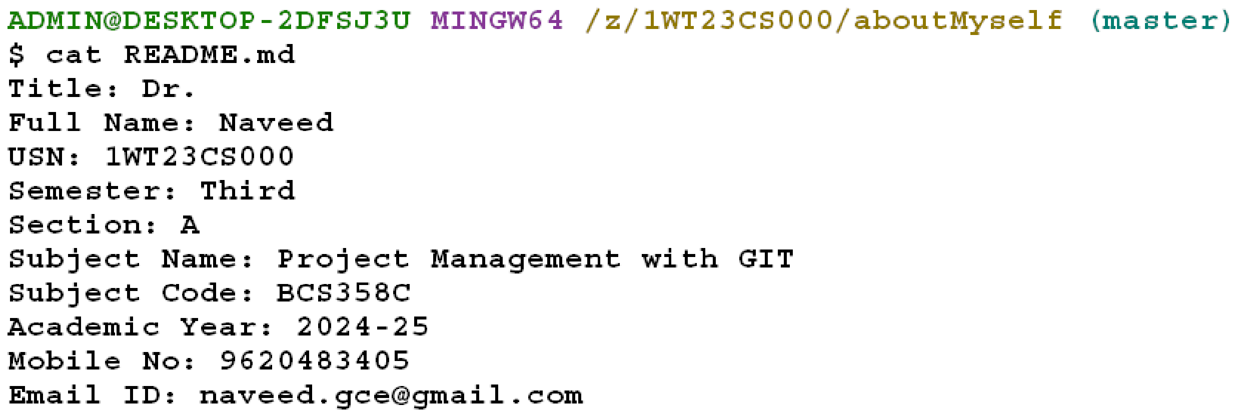
```
MINGW64:/z/1WT23CS000/aboutMyself
  GNU nano 8.1                                                README.md
Title: Dr.
Full Name: Naveed
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with GIT
Subject Code: BCS358C
Academic Year: 2024-25
Mobile No: 9620483405
Email ID: naveed.gce@gmail.com
```

To save:Ctrl+S
To Exit: Ctrl+X
To check the contents of the README.md File



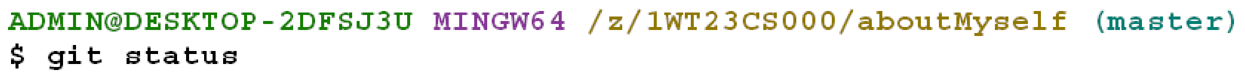
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat README.md
```

It will highlight the contents of the file



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat README.md
Title: Dr.
Full Name: Naveed
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with GIT
Subject Code: BCS358C
Academic Year: 2024-25
Mobile No: 9620483405
Email ID: naveed.gce@gmail.com
```

To edit or modify the contents of the README.md file the same above procedure may be adopted.
The data will be updated. But it will not be committed in the git. If we check the status:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git status
```

It will show changes to be committed with a new file README.md highlighted in green color.

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md

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
```

To commit the above with a message “Added README.md file with basic data successfully”

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git add .
```

And then use

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git commit -m "Added README.md file with basic data successfully"
```

It will show you the message:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git commit -m "Added README.md file with basic data successfully"
[master (root-commit) 4dda3ea] Added README.md file with basic data successfully
 1 file changed, 11 insertions(+)
 create mode 100644 README.md
```

If you want to check the status:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git status
```

It will show you nothing to commit, working tree clean:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git status
On branch master
nothing to commit, working tree clean
```

`git add .` is used in Git to stage all changes in the current directory and its subdirectories for the next commit. When you make changes to files in your Git repository, these changes are initially considered "unstaged" or "untracked." Staging files with `git add .` prepares these changes to be included in the next commit snapshot of your project.
Here’s a breakdown of what `git add .` does:
Staging Changes: It adds all modified (tracked) files and all new files (untracked) to the staging area.
Recursive Operation: The `.` represents the current directory and its subdirectories, so `git add .` recursively adds changes from all directories and subdirectories.
Efficiency: It's a convenient shortcut to stage multiple files and changes quickly without specifying each file individually.
After staging changes with `git add .`, you typically follow up with `git commit` to permanently store those changes in the Git repository history.

OUTPUT:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat README.md
Title: Dr.
Full Name: Naveed
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with GIT
Subject Code: BCS358C
Academic Year: 2024-25
Mobile No: 9620483405
Email ID: naveed.gce@gmail.com
```

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Experiment_03 Creating and Managing Branches

Aim:

1. To create a new branch named "feature-branch”.
2. To add a text file “aboutMyself.txt” into the repository /1WT23CS000/aboutMyself .
3. To add the contents into the text file given below:
    - `Title: Dr.`
    - `Full Name: Naveed`
    - `USN: 1WT23CS000`
    - `Semester: Third`
    - `Section: A`
    - `Branch: Mechanical Engineering`
    - `Year of Admission: 2023`
    - `College Name: GITW`
4. To commit a message “Added text file aboutMyself.txt successfully”.

Normally the default branch will be master branch or in some system it will be main branch. During the development stage it is desirable to create a new branch with any name and add all the required files in it and carry on the process. Whatever the files added into the master branch it will be visible in the new branch.But if we create a new file in new branch such as feature-branch it will not be highlighted in master branch. Even if we modify the content of the file of master branch in the new feature-branch it will not be updated until merged.

To create a new branch “feature-branch”:

First open the repository by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /
$ cd /z/1WT23CS000/aboutMyself
```

It will show:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
```

By default, it will be master branch or in some case it will be main branch. To check out which branch the current repository belongs use:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch
```

It will show

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch
* master
```

Though the type of branch will be normally highlighted under () in the drive itself.But still it is required to checkout, the above can be used.

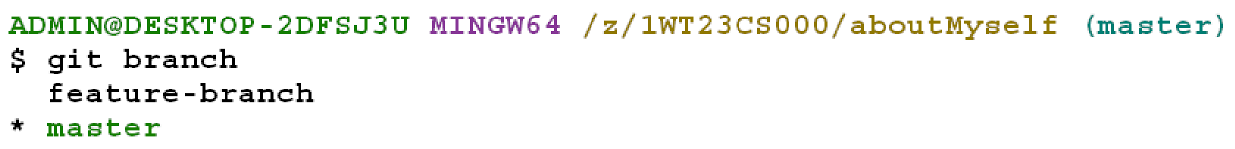
To create a new branch “feature-branch”:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch feature-branch
```

Let us check whether the branch is created or not by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch
```

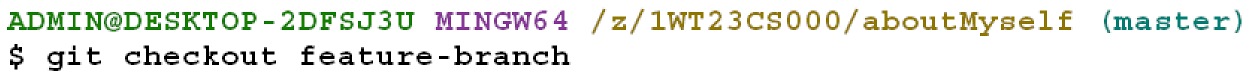
It will show:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch
  feature-branch
* master
```

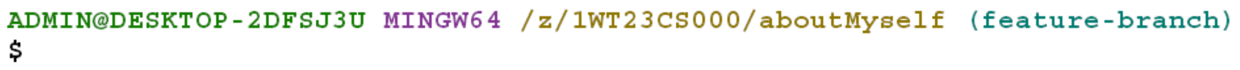
Now we can see there are two branches.It will highlight the number of branches available with the active branch shown in green color with * symbol

To shift to the new branch created:



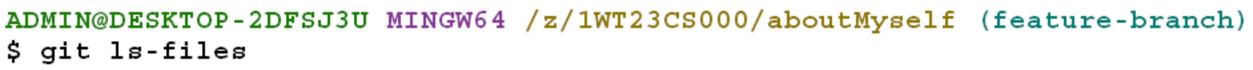
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git checkout feature-branch
```

It will shift to the new branch



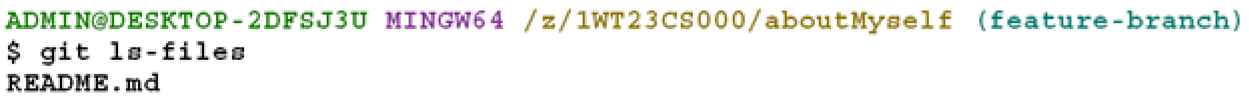
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$
```

If we check the number of files available in feature-branch by using



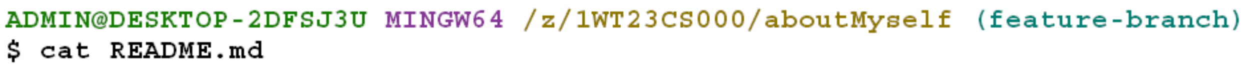
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git ls-files
```

It shows:



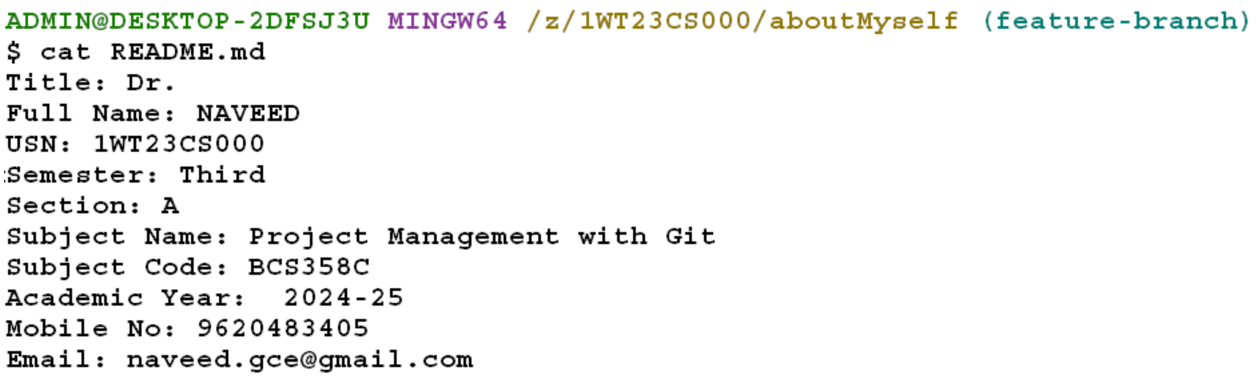
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git ls-files
README.md
```

It is shows README.md file which was created in master branch.If the check the contents of the README.md file by using:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat README.md
```

It shows:



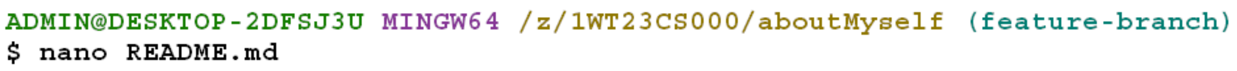
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat README.md
Title: Dr.
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with Git
Subject Code: BCS358C
Academic Year:  2024-25
Mobile No: 9620483405
Email: naveed.gce@gmail.com
```

Therefore, whatever files that were created in master branch are also available in the feature-branch with the same contents.
But if we modify the contents of README.md file in feature-branch, will it get reflected in master branch?

/**********/

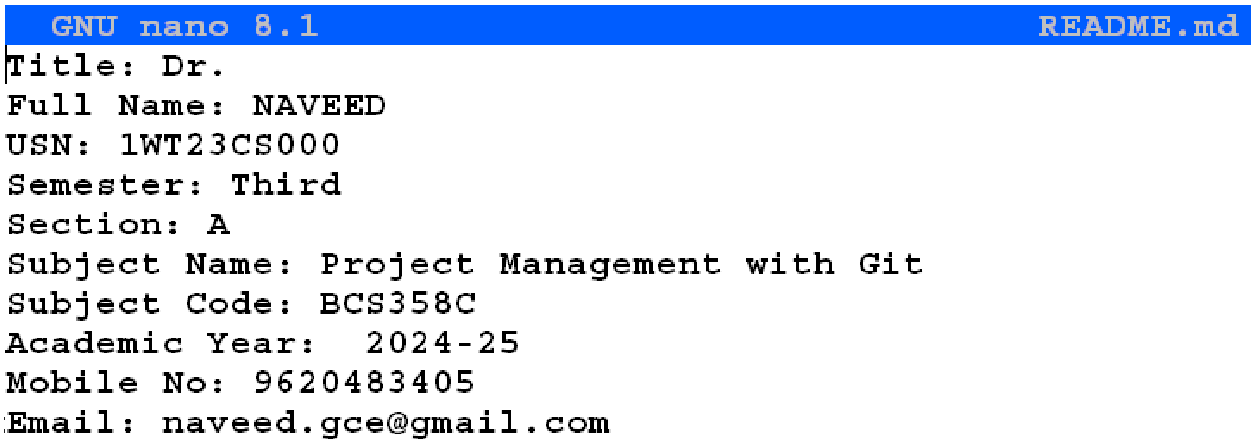
/* For Knowledge Purpose Not for Record Writing */

Let us checkout.
Remove Email ID from the file README.md available in feature-branch by using:



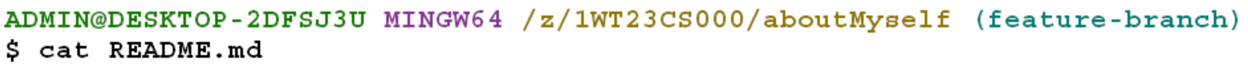
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ nano README.md
```

It shows:



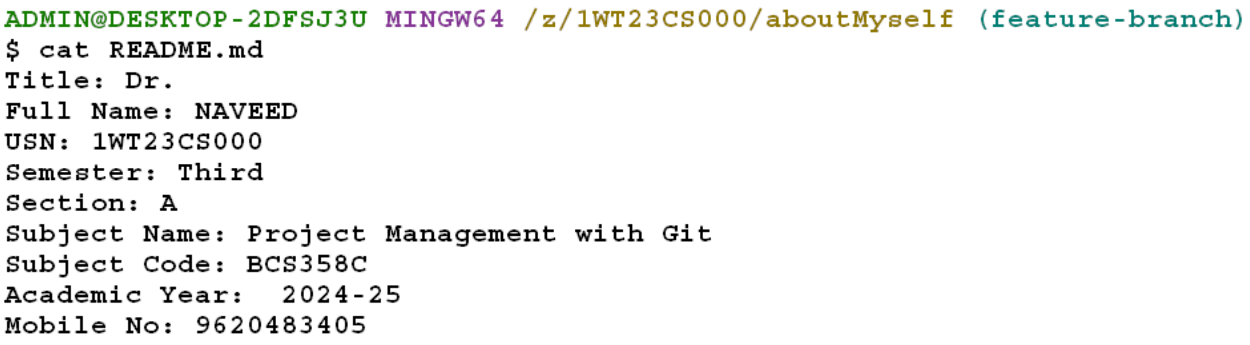
```
  GNU nano 8.1                                                README.md
Title: Dr.
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with Git
Subject Code: BCS358C
Academic Year:  2024-25
Mobile No: 9620483405
Email: naveed.gce@gmail.com
```

Remove Email from the above and save the file.
Again, let us check the content by using:



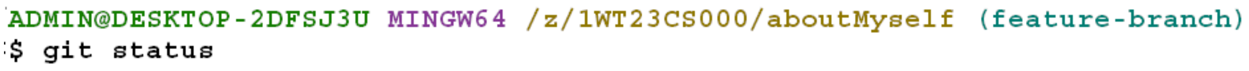
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat README.md
```

It shows:



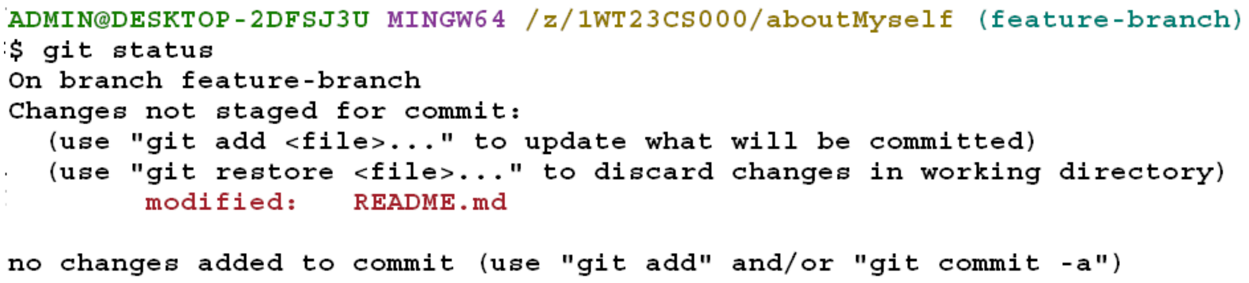
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat README.md
Title: Dr.
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with Git
Subject Code: BCS358C
Academic Year:  2024-25
Mobile No: 9620483405
```

Therefore, there is no Email present in the above content. Now if we check the status by using:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git status
```

It shows:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git status
On branch feature-branch
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

no changes added to commit (use "git add" and/or "git commit -a")
```

It tells us to save it with committed message. Let us save by committing a message "Removed Email Successfully from README.md"

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git add .

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git commit -m "Removed Email successfully from README.md file"
[feature-branch 7c1af39] Removed Email successfully from README.md file
 1 file changed, 1 insertion(+), 1 deletion(-)

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git status
On branch feature-branch
nothing to commit, working tree clean
```

Now let us switch back to the master branch and check the contents of the same file README.md by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git checkout master
```

It will get switched to master branch

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
```

Now let us check the contents of README.md file by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat README.md
```

It shows:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat README.md
Title: Dr.
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with Git
Subject Code: BCS358C
Academic Year:  2024-25
Mobile No: 9620483405
Email: naveed.gce@gmail.com
```

We can see that the Email which was deleted in feature-branch is not deleted in master branch. Therefore, if we want to update it into the master branch as well then git-merging will be used.

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git merge feature-branch
```

It shows

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git merge feature-branch
Updating d9068ac..7c1af39
Fast-forward
 README.md | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)
```

Now if we check the contents of README.md file in master branch by using

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat README.md
```

It shows:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat README.md
Title: Dr.
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with Git
Subject Code: BCS358C
Academic Year:  2024-25
Mobile No: 9620483405
```

Now we can see that after merging the contents are updated and we don’t find Email in the above content.

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git add .

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git commit -m "Merged feature-branch into the master branch with updated contents"
On branch master
nothing to commit, working tree clean
```

/*********/

To Create a new text file by name “aboutMyself.txt”

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git add aboutMyself.txt
```

It shows some error.

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git add aboutMyself.txt
fatal: pathspec 'aboutMyself.txt' did not match any files
```

To solve the error, let us add an untraced file by using

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ echo > aboutMyself.txt
```

Then we will use

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git add aboutMyself.txt
```

It will show some warning related to operating system. Just ignore it.
Now if we check the number of files available:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git ls-files
```

It shows two files available.

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git ls-files
README.md
aboutMyself.txt
```

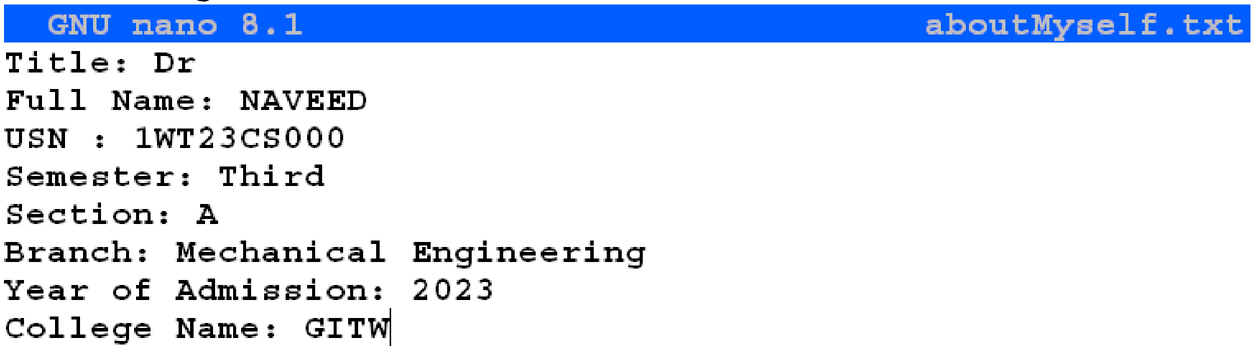
Add the contents by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ nano aboutMyself.txt
```

A file will open, type the contents and save with Ctrl+S and exit by using Ctrl+X.

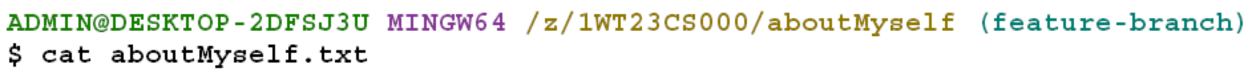
- `Title: Dr.`
- `Full Name: Naveed`

- USN: 1WT23CS000
- Semester: Third
- Section: A
- Branch: Mechanical Engineering
- Year of Admission: 2023
- College Name: GITW



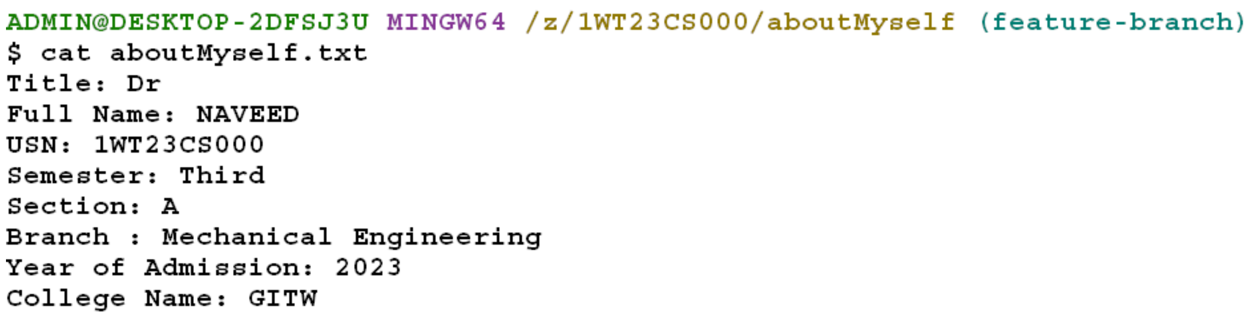
```
  GNU nano 8.1                                        aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN : 1WT23CS000
Semester: Third
Section: A
Branch: Mechanical Engineering
Year of Admission: 2023
College Name: GITW
```

To check the contents of the file use:



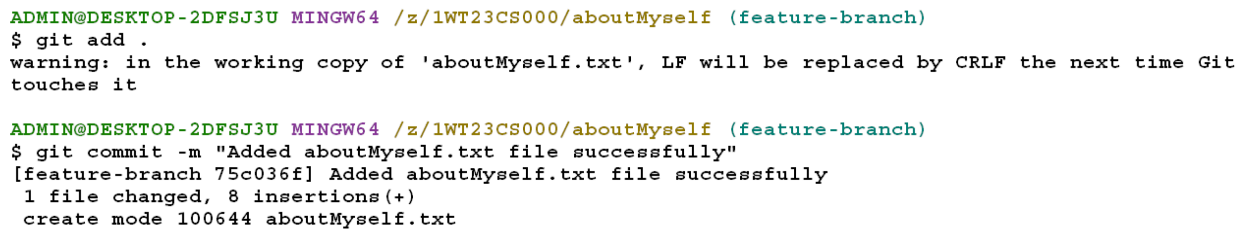
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat aboutMyself.txt
```

It shows:



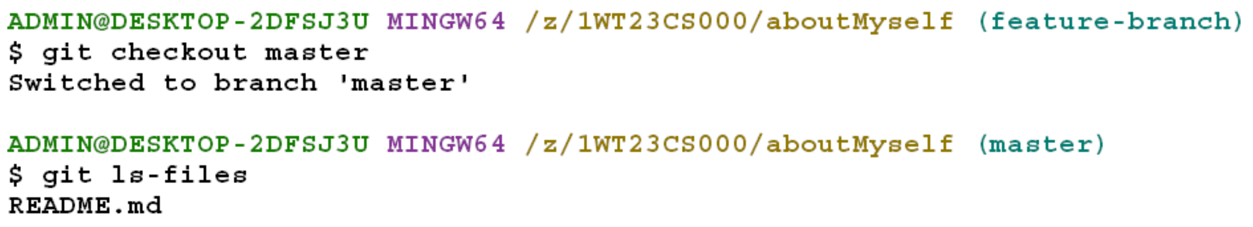
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
```

Save the file by committing a message “Added aboutMyself.txt file successfully”



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git add .
warning: in the working copy of 'aboutMyself.txt', LF will be replaced by CRLF the next time Git
touches it

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git commit -m "Added aboutMyself.txt file successfully"
[feature-branch 75c036f] Added aboutMyself.txt file successfully
 1 file changed, 8 insertions(+)
 create mode 100644 aboutMyself.txt
```

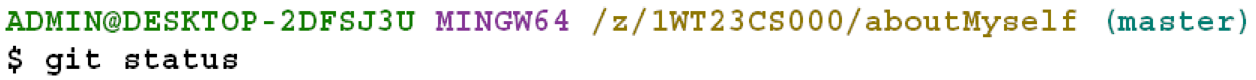
Now let us find whether the same file is added into the master branch with same contents.



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git checkout master
Switched to branch 'master'

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git ls-files
README.md
```

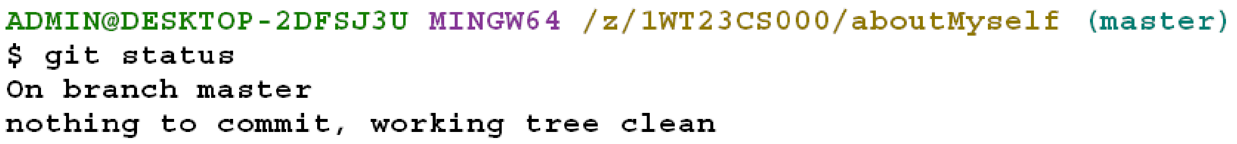
Here we don’t find the file aboutMyself.txt which was created in feature-branch.

If we check the status,



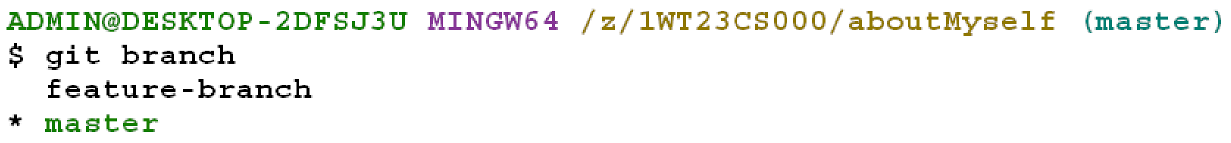
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git status
```

It will show a message of working tree clean:



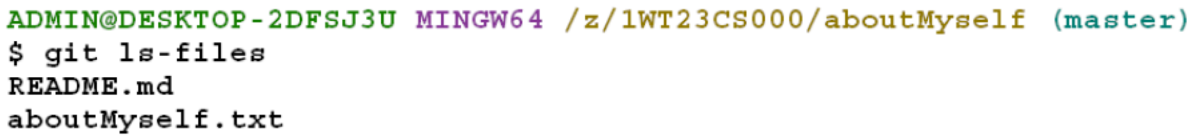
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git status
On branch master
nothing to commit, working tree clean
```

OUTPUT:

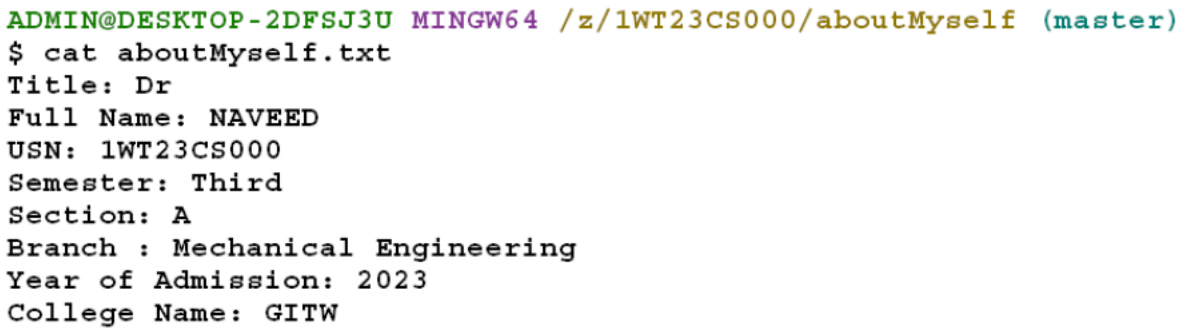


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch
  feature-branch
* master
```

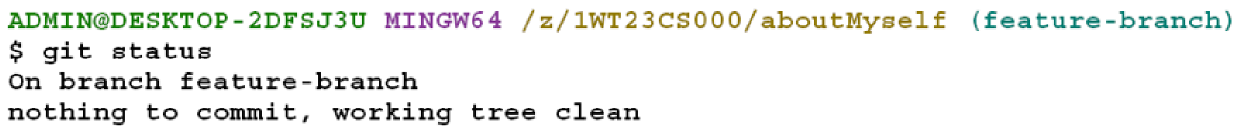
OUTPUT:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git ls-files
README.md
aboutMyself.txt
```



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
```



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git status
On branch feature-branch
nothing to commit, working tree clean
```

Experiment-04: Merging of Feature-Branch into the Master Branch:

Aim:

To merge feature-branch into the master branch

Now let us find whether the same file is added into the master branch with same contents.

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git checkout master
Switched to branch 'master'

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git ls-files
README.md
```

Here we don't find the file aboutMyself.txt which was created in feature-branch.

To make the file aboutMyself.txt available in master branch git merging is used:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git merge feature-branch
```

It shows:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git merge feature-branch
Updating 7c1af39..75c036f
Fast-forward
 aboutMyself.txt | 8 ++++++++
 1 file changed, 8 insertions(+)
 create mode 100644 aboutMyself.txt
```

Now if we check the files available in master branch by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git ls-files
```

It shows two files.

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git ls-files
README.md
aboutMyself.txt
```

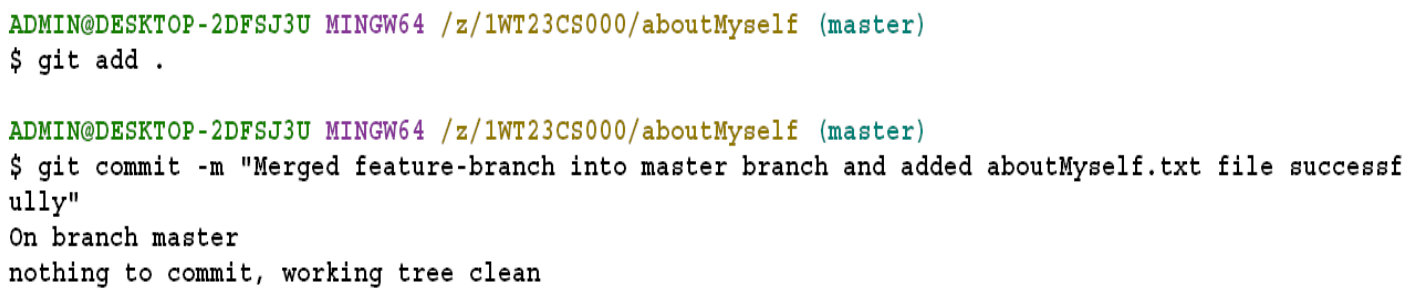
Now if we check the contents of aboutMyself.txt file by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat aboutMyself.txt
```

It shows:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
```

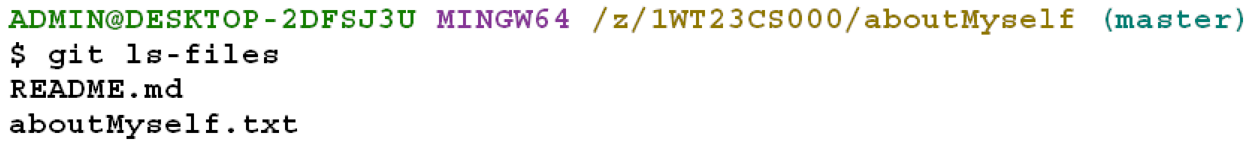
Commit the above with a message by using:



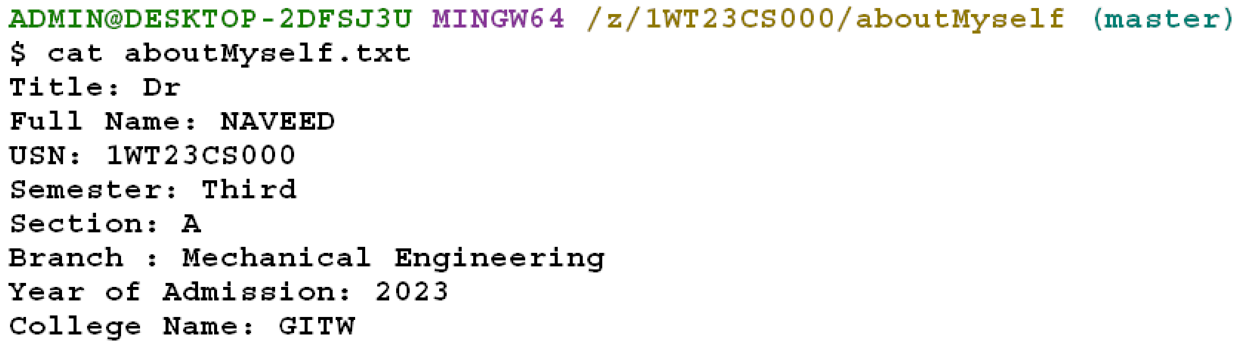
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git add .

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git commit -m "Merged feature-branch into master branch and added aboutMyself.txt file successf
ully"
On branch master
nothing to commit, working tree clean
```

Merging feature branches into the main branch is a fundamental practice in modern software development. It ensures that new features and improvements are regularly integrated, tested, and made available to all team members, leading to a more robust and maintainable codebase. By following best practices, teams can effectively manage their development process and deliver high-quality software.

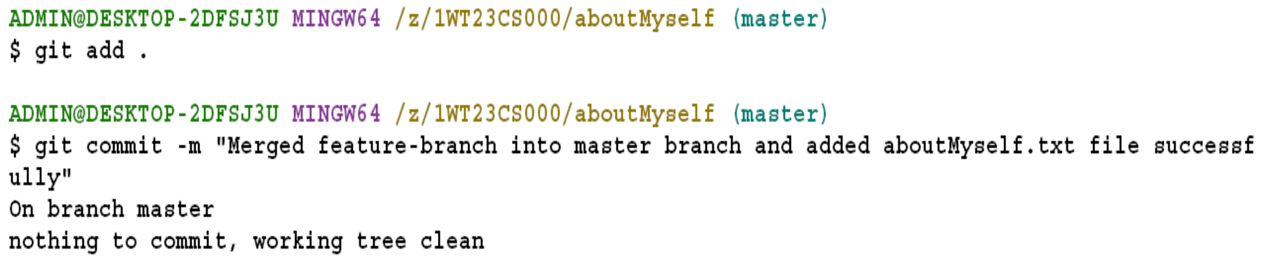
OUTPUT:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git ls-files
README.md
aboutMyself.txt
```



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
```



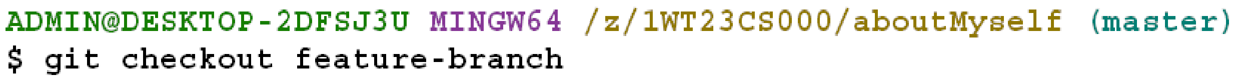
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git add .

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git commit -m "Merged feature-branch into master branch and added aboutMyself.txt file successf
ully"
On branch master
nothing to commit, working tree clean
```

Experiment-05: Updating of files in the Feature-Branch

Aim:

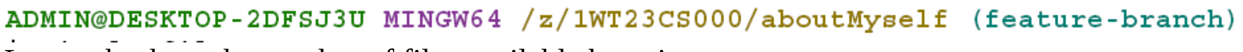
1. To add the Qualification Details given Below into the file “aboutMyself.txt” of the feature-branch:
   UG Degree: B.E – Mechanical Engineering
   PG Degree:M.Tech – Manufacturing
   PhD Degree: Materials Science
2. To commit the above with a message “Added Qualification Details Successfully”
3. To merge the feature-branch with the master branch and commit with a message “Merged the feature-branch Successfully and updated the file with Qualification Details”.
4. To delete the feature-branch if no longer needed.
5. To get back the deleted feature-branch

Before proceeding checkout to the feature-branch from master branch by using:



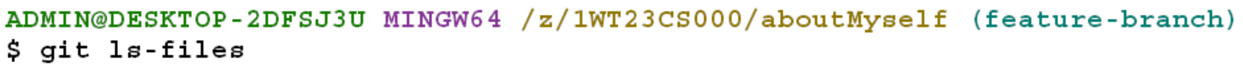
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git checkout feature-branch
```

It will get shifted to feature-branch



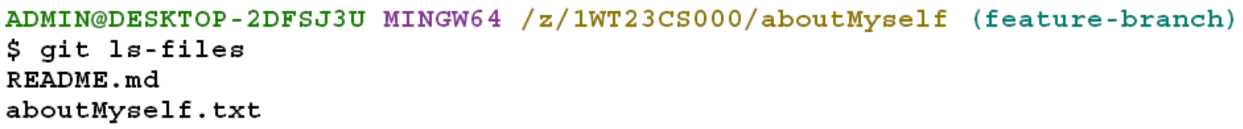
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
```

Let us checkout the number of files available by using:



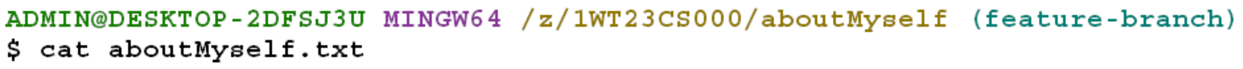
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git ls-files
```

It shows two files:



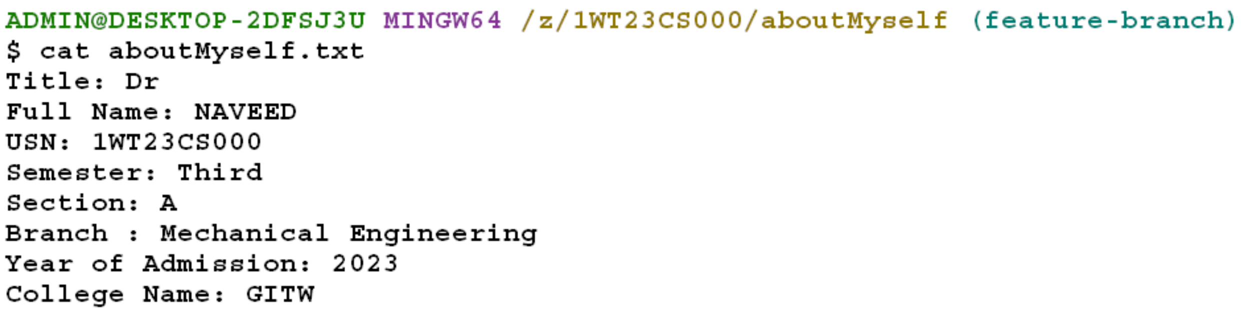
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git ls-files
README.md
aboutMyself.txt
```

Now let us checkout the contents of the file “aboutMyself.txt ” by using:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat aboutMyself.txt
```

It shows:

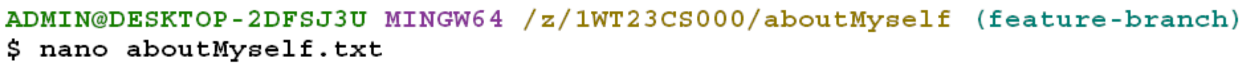


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
```

To the above add the following data:

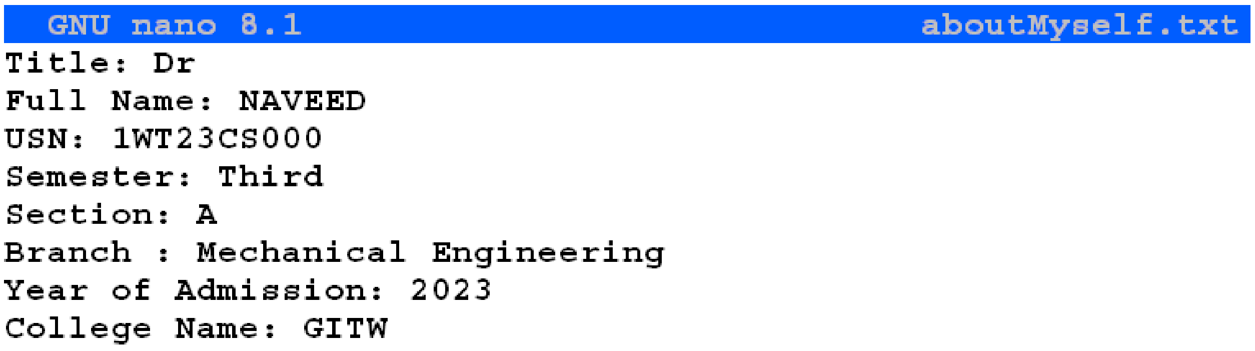
UG Degree: B.E – Mechanical Engineering
PG Degree:M.Tech – Manufacturing
PhD Degree: Materials Science

By using:



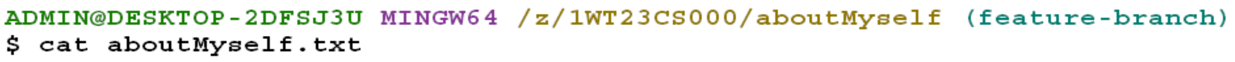
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ nano aboutMyself.txt
```

It will open a GNU file window as shown below:



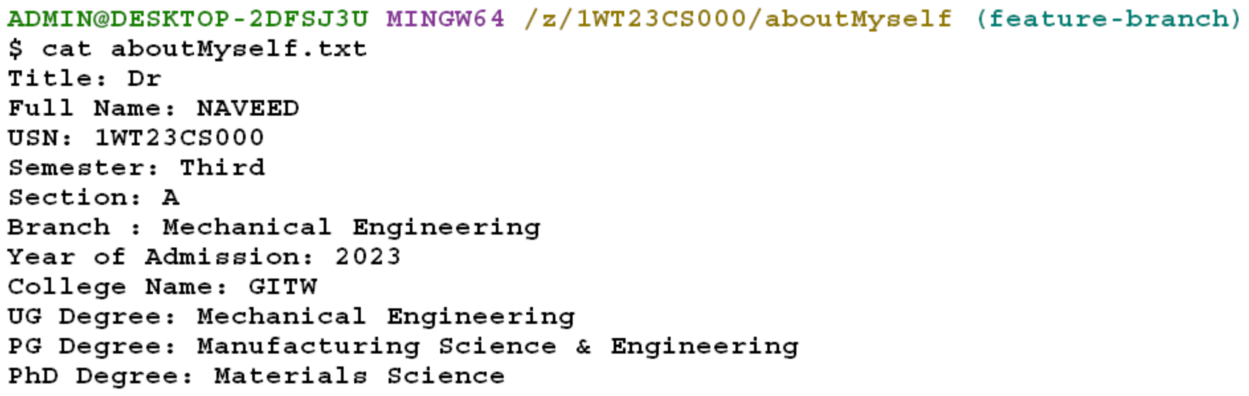
```
  GNU nano 8.1                                    aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
```

Now type the above data. Save the data with Ctrl+S and exit the screen using Ctrl+X
Now let us check the data



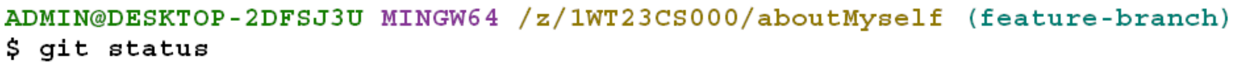
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat aboutMyself.txt
```

It shows:



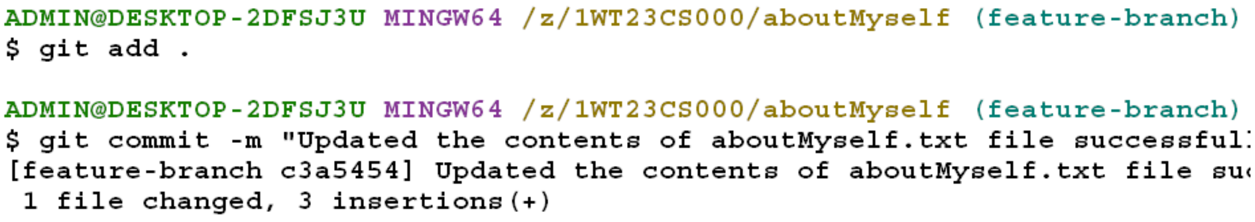
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
UG Degree: Mechanical Engineering
PG Degree: Manufacturing Science & Engineering
PhD Degree: Materials Science
```

We can see the data is updated. If we check the status by using:



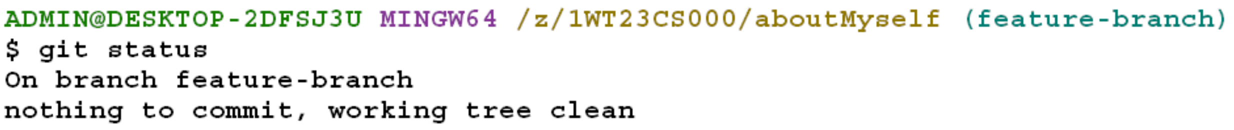
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git status
```

It shows that the file is modified but it is not committed. added. To commit with a message. To commit with a message use:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git add .

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git commit -m "Updated the contents of aboutMyself.txt file successful
[feature-branch c3a5454] Updated the contents of aboutMyself.txt file su
 1 file changed, 3 insertions(+)
```

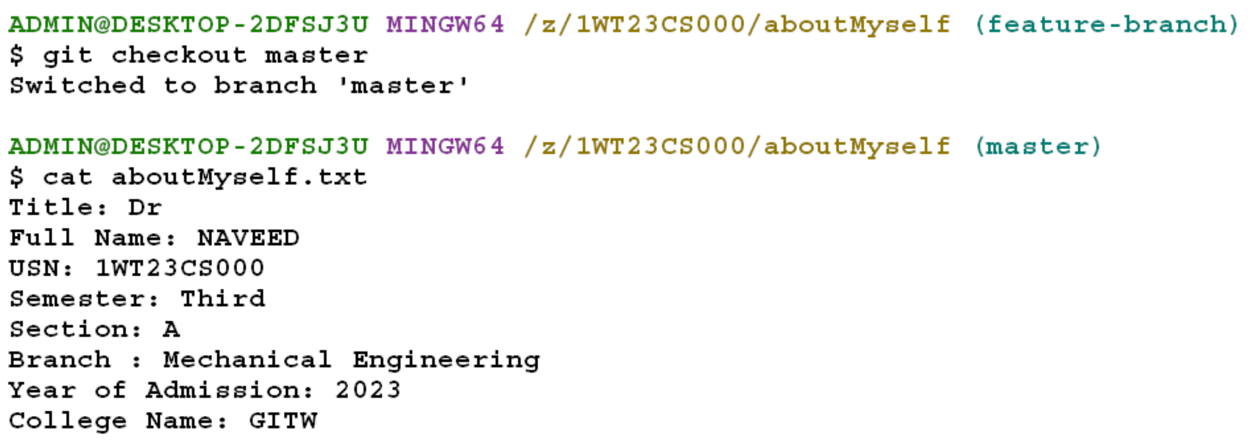
Now if we check the status:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git status
On branch feature-branch
nothing to commit, working tree clean
```

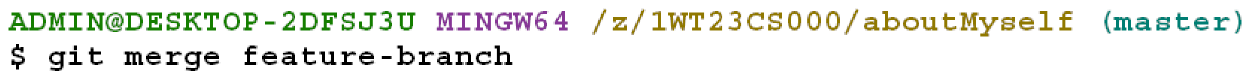
The working tree is clean.

Now let us check whether in master branch the data is updated or not. We will shift to the master branch and checkout the contents of the file aboutMyself.txt



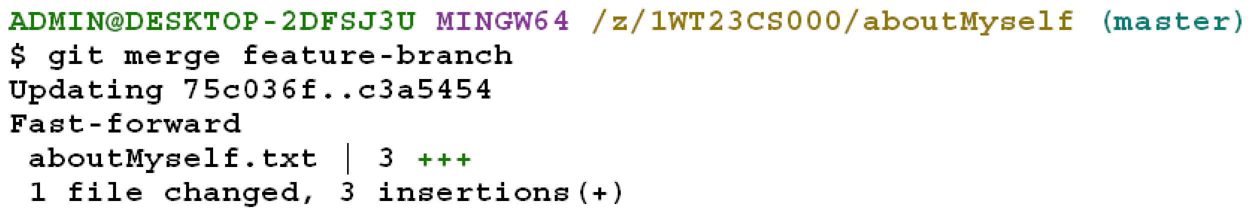
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git checkout master
Switched to branch 'master'

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
```

We can see that the contents of the file aboutMyself.txt are not updated in the master branch. To update the contents git-merging will be used as shown below:



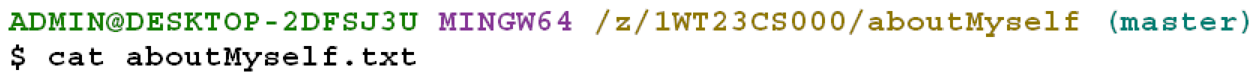
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git merge feature-branch
```

It will merge the contents of the file aboutMyself.txt present in feature-branch. It shows:



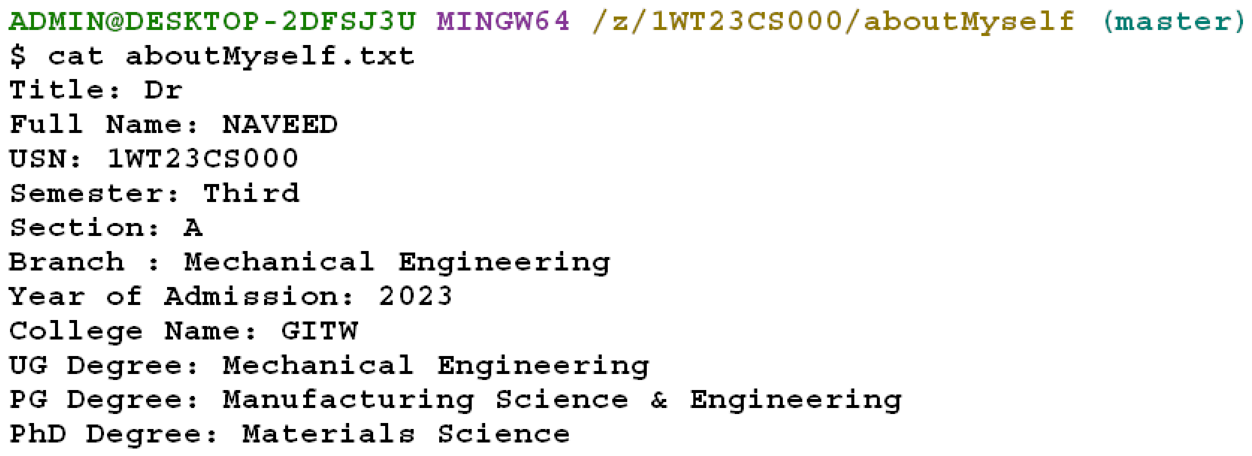
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git merge feature-branch
Updating 75c036f..c3a5454
Fast-forward
 aboutMyself.txt | 3 +++
 1 file changed, 3 insertions(+)
```

Now if we check the contents



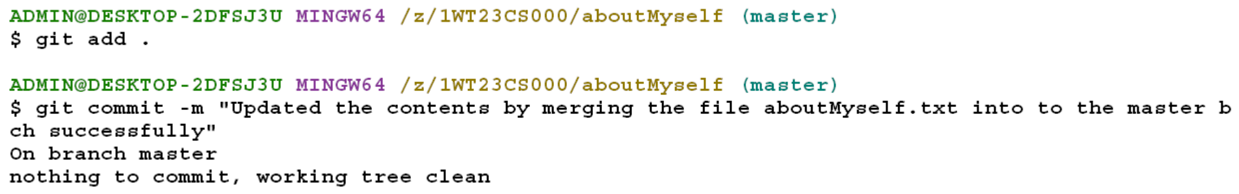
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat aboutMyself.txt
```

It will show:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
UG Degree: Mechanical Engineering
PG Degree: Manufacturing Science & Engineering
PhD Degree: Materials Science
```

Now we can see that the contents are now updated after merging.



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git add .

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git commit -m "Updated the contents by merging the file aboutMyself.txt into to the master b
ch successfully"
On branch master
nothing to commit, working tree clean
```

Note: If it shows some conflict resolve the issue and then proceed.

To delete feature-branch:

First, we will check no. of branches available by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch
  feature-branch
* master
```

It shows two branches with master branch in active mode.
To delete we will use:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch -d feature-branch
```

It will show:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch -d feature-branch
Deleted branch feature-branch (was c3a5454).
```

If we check the number of branches available by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch
```

It will show:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch
* master
```

It shows only one branch i.e., master branch.
To get back the feature-branch:
To get back the deleted feature-branch use:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git checkout -b feature-branch
```

It will restore and get back to the feature-branch:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git checkout -b feature-branch
Switched to a new branch 'feature-branch'
```

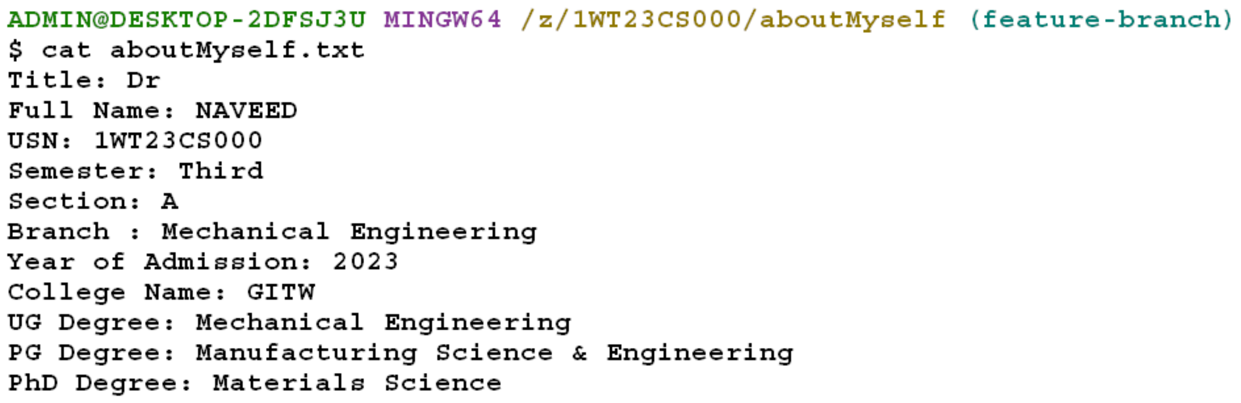
Now if we check the number of branches available:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git branch
* feature-branch
  master
```

It shows two branches. Now let us check the number of files available in feature-branch

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git ls-files
README.md
aboutMyself.txt
```

It shows the same two files that were available before deleting the branch. Let us check the contents of aboutMysellf.txt file:

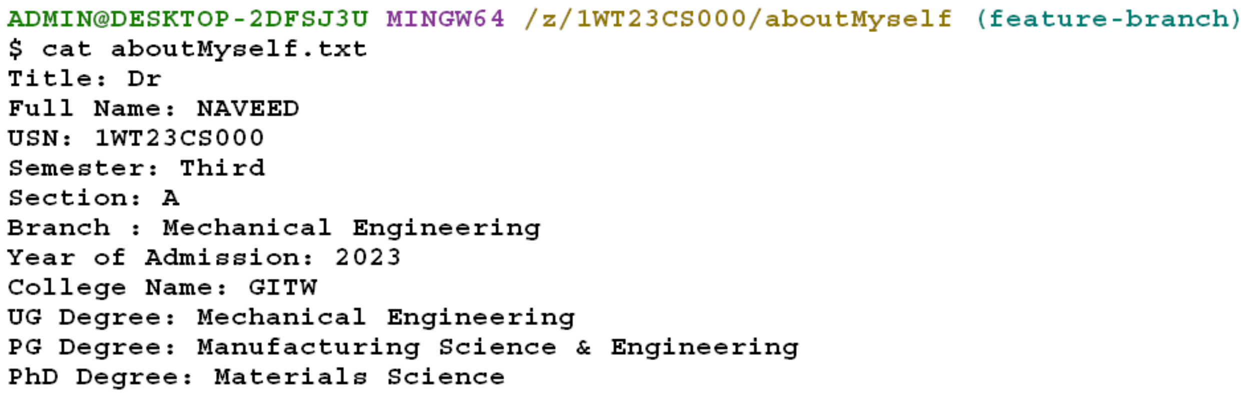


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
UG Degree: Mechanical Engineering
PG Degree: Manufacturing Science & Engineering
PhD Degree: Materials Science
```

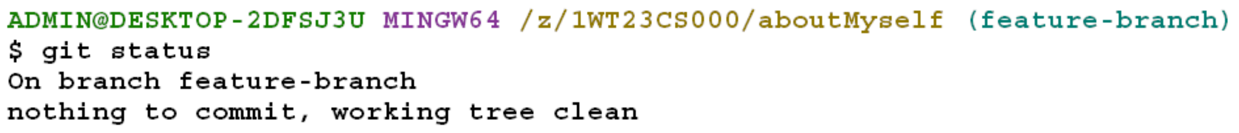
It shows the same contents which were available before deleting.

Note: Always download the updated version of git bash. In some version it will show some reference code when "git reflog" command is used. Then to restore the deleted branch "git checkout -b feature-branch ref. code" command will be used.

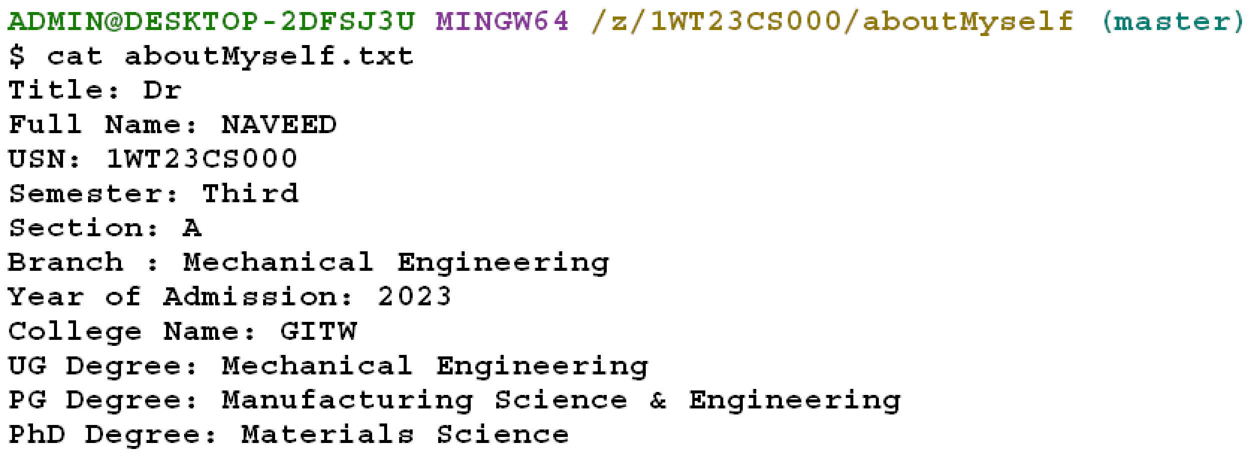
OUTPUT:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
UG Degree: Mechanical Engineering
PG Degree: Manufacturing Science & Engineering
PhD Degree: Materials Science
```



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git status
On branch feature-branch
nothing to commit, working tree clean
```



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
UG Degree: Mechanical Engineering
PG Degree: Manufacturing Science & Engineering
PhD Degree: Materials Science
```

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git add .

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git commit -m "Updated the contents by merging the file aboutMyself.txt into to the master b
ch successfully"
On branch master
nothing to commit, working tree clean
```

Experiment-06: Light Weight and Annotated Tags

Aim:

1. To update the file "README.md" and add "Date of Joining to GITW: 1st Oct-2023" to it under the repository /1WT23CS000/aboutMyself under master branch.
2. To commit with a message "My Date of Joining to GITW Updated successfully"
3. To add a light weight tag v1.0 into file "README.md"
4. To add an Annotated tag v2.0 with a message "My Date of Joining to GITW" into the same file

In Git, there are two main types of tags:

1. **Lightweight Tags**
2. **Annotated Tags**

Each type serves different purposes and has different characteristics.

1. Lightweight Tags

Lightweight tags are simple and act like a pointer to a specific commit. They are just a name (like a branch) that points to a specific commit. Lightweight tags do not contain any additional metadata such as the tagger name, date, or a tagging message.

Characteristics:

- Simple and fast to create.
- Do not store any additional information beyond the commit they point to.
- Similar to a branch that doesn't change.

Lightweight tags are useful when you just want to mark a specific point in your history, such as a commit representing a release, without the need for additional metadata.

2. Annotated Tags

Annotated tags store additional metadata, including the tagger's name, email, date, and a tagging message. They are stored as full objects in the Git database, which makes them more robust and verifiable.

Characteristics:

- Include metadata (tagger name, email, date, and message).
- Stored as full objects in the Git database.
- Can be signed with GPG to verify authenticity.

Annotated tags are suitable for marking releases or other significant milestones where you want to include detailed information about the tag.

Summary

- **Lightweight Tags**: Simple, fast, and just a pointer to a commit. Created with git tag tagname.
- **Annotated Tags**: Contain metadata, are stored as full objects, and can be signed. Created with git tag -a tagname -m "message".

Use lightweight tags for simple markers and annotated tags when you need more information and robustness.

First ensure that we are in master branch. Let us checkout the number of files available in master branch by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git ls-files
```

It will show:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git ls-files
README.md
aboutMyself.txt
```

It shows two files. Let us checkout the contents of the README.md file by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat README.md
```
It shows:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat README.md
Title: Dr.
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with Git
Subject Code: BCS358C
Academic Year:  2024-25
Mobile No: 9620483405
```
Now to the above we will add a content such as “My date of Joining to GITW” by using

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ nano README.md
```
It will open GNU screen. Add the data and save it and exit by using Ctrl+S and Ctrl+X.

```
  GNU nano 8.1                                        README.md
Title: Dr.
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with Git
Subject Code: BCS358C
Academic Year:  2024-25
Mobile No: 9620483405
My Date of Joining to GITW: 3rd Oct-2023
```
Now if we check the contents.

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat README.md
Title: Dr.
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with Git
Subject Code: BCS358C
Academic Year:  2024-25
Mobile No: 9620483405
My Date of Joining to GITW: 3rd Oct-2023
```
We can see the contents are updated.
Now we will commit the above with a message “My Date of Joining to GITW Updatedsuccessfully” by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git add .

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git commit -m "My Date of Joinining to GITW updated successfully"
[master 44fcdcb] My Date of Joinining to GITW updated successfully
 1 file changed, 1 insertion(+), 1 deletion(-)
```

To add lightweight Tag v1.0 to the above committed message:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git tag v1.0
```

It will add the tag by name v1.0. Now to check whether it is added or not use:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git tag
```

It will show all the tags available

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git tag
v1.0
```

It shows one tag by name v1.0. Now let us check the contents of the tag v1.0 by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git show v1.0
```

It will show:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git show v1.0
commit 44fcdcb8b4cbd09c285b662173c7cd65eefc0ecf (HEAD -> master, tag: v1.0)
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Tue Aug 27 14:40:07 2024 +0500

    My Date of Joinining to GITW updated successfully

diff --git a/README.md b/README.md
index 3add72c..4802ed1 100644
--- a/README.md
+++ b/README.md
@@ -7,4 +7,4 @@ Subject Name: Project Management with Git
 Subject Code: BCS358C
 Academic Year:  2024-25
 Mobile No: 9620483405
-
+My Date of Joining to GITW: 3rd Oct-2023
```

We can see the committed message along with the contents added to the above file in last line with green color.

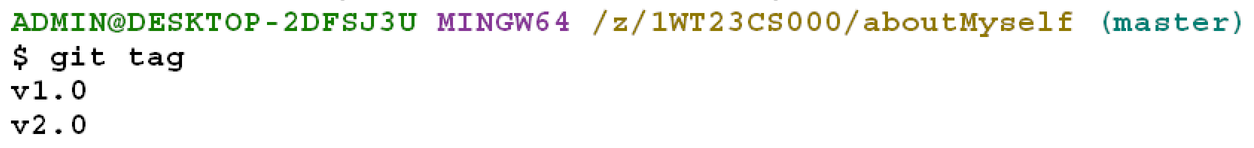
Light wight tags will show only the committed message. It will not show the tagger name and its details. For those Annotated tags are used as shown below:

To create Annotated tag v2.0 with a message "My Date of Joining to GITW"

Use the following code:

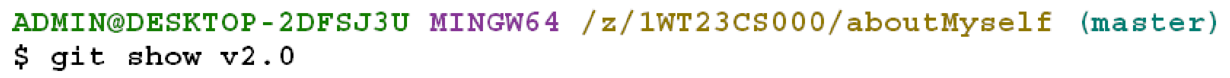
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git tag -a v2.0 -m "My DAte of Joining to GITW"
```

It will add an annotated tag v2.0. To check the number of tags available use:



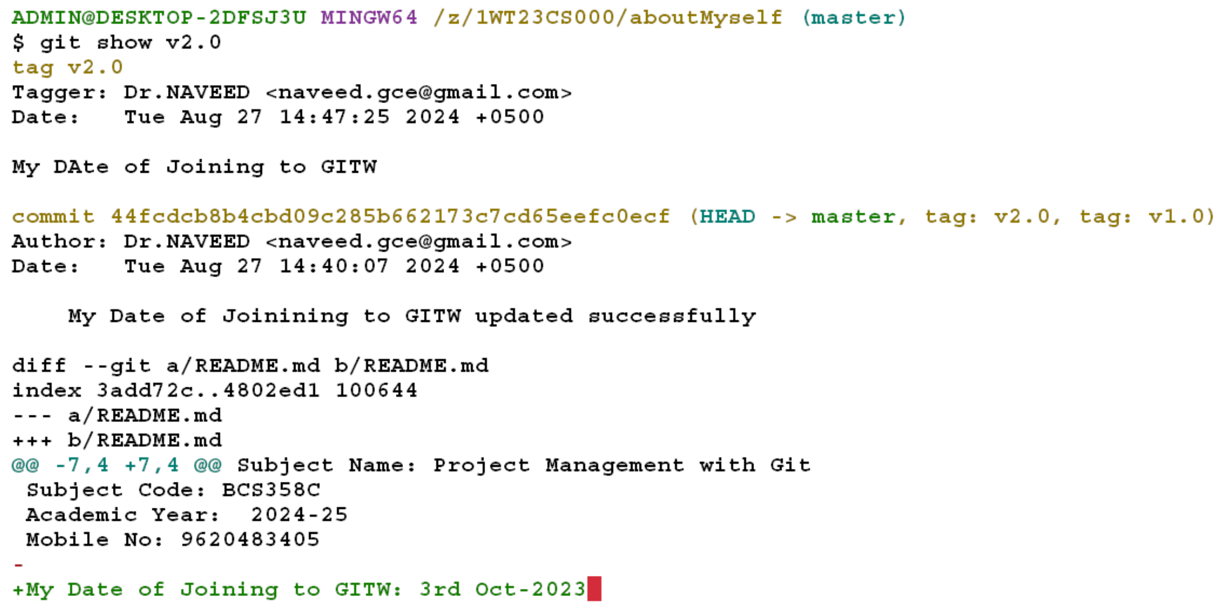
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git tag
v1.0
v2.0
```

It shows two tags. To check the contents of tag v2.0 use:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git show v2.0
```

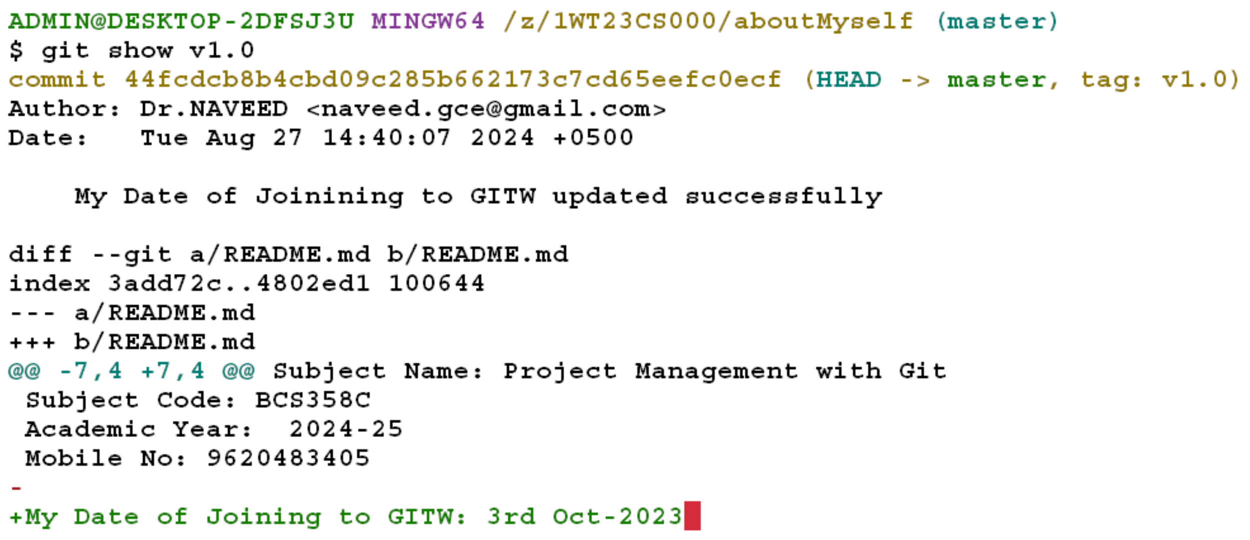
It will show:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git show v2.0
tag v2.0
Tagger: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Tue Aug 27 14:47:25 2024 +0500

My DAte of Joining to GITW

commit 44fcdcb8b4cbd09c285b662173c7cd65eefc0ecf (HEAD -> master, tag: v2.0, tag: v1.0)
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Tue Aug 27 14:40:07 2024 +0500

    My Date of Joinining to GITW updated successfully

diff --git a/README.md b/README.md
index 3add72c..4802ed1 100644
--- a/README.md
+++ b/README.md
@@ -7,4 +7,4 @@ Subject Name: Project Management with Git
 Subject Code: BCS358C
 Academic Year:  2024-25
 Mobile No: 9620483405
-
+My Date of Joining to GITW: 3rd Oct-2023
```

It will show details with tagger name and its details:

OUTPUT:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git show v1.0
commit 44fcdcb8b4cbd09c285b662173c7cd65eefc0ecf (HEAD -> master, tag: v1.0)
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Tue Aug 27 14:40:07 2024 +0500

    My Date of Joinining to GITW updated successfully

diff --git a/README.md b/README.md
index 3add72c..4802ed1 100644
--- a/README.md
+++ b/README.md
@@ -7,4 +7,4 @@ Subject Name: Project Management with Git
 Subject Code: BCS358C
 Academic Year:  2024-25
 Mobile No: 9620483405
-
+My Date of Joining to GITW: 3rd Oct-2023
```

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git show v2.0
tag v2.0
Tagger: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Tue Aug 27 14:47:25 2024 +0500

My DAte of Joining to GITW

commit 44fcdcb8b4cbd09c285b662173c7cd65eefc0ecf (HEAD -> master, tag: v2.0, tag: v1.0)
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Tue Aug 27 14:40:07 2024 +0500

    My Date of Joinining to GITW updated successfully

diff --git a/README.md b/README.md
index 3add72c..4802ed1 100644
--- a/README.md
+++ b/README.md
@@ -7,4 +7,4 @@ Subject Name: Project Management with Git
 Subject Code: BCS358C
 Academic Year:  2024-25
 Mobile No: 9620483405
-
+My Date of Joining to GITW: 3rd Oct-2023
```

Experiment-07: Analyzing GIT History

Aim:

1. To view the details of specific commit, including the author, date, and commit messageforthe givencommit ID.
2. To display the commits made in master branch
3. To obtain details of the commit with full details such as Author name, Email, date, timings, committed messagefor the given ID 44fcdcbof master branch
4. To display the commits made in feature-branch
5. To obtain details of the commit with full details such as Author name, Email, date, timings, committed message for the given IDc3a5454of feature-branch
6. To write the command to list all commits made by the author "Dr. NAVEED" between "2024-01-01"and "2024-12-31."
7. To write the command to display the last five commits in the repository's history for master branch as well feature-branch.

First find out how many branches are available or how many branches were created other than the default master branch by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch
```

It will show the available branches with current branch in green color.

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch
  feature-branch
* master
```

To display commits made in master branch use:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master --oneline
```

It will show:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master --oneline
44fcdcb (HEAD -> master, tag: v2.0, tag: v1.0) My Date of Joinining to GITW updated successfully
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
ac80e2e Removed Email ID
d9068ac Added README.md file successfully
4dda3ea Added README.md file with basic data successfully
```

Note: Running the git log master --oneline command will show you a simplified, one-line-per-commit log of the feature-branch. This is useful for quickly viewing the commit history in a compact format.

To check committed details for the given ID44fcdcb:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git show 44fcdcb
```

It will show:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git show 44fcdcb
commit 44fcdcb8b4cbd09c285b662173c7cd65eefc0ecf (HEAD -> master, tag: v2.0, tag: v1.0)
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Tue Aug 27 14:40:07 2024 +0500

    My Date of Joinining to GITW updated successfully

diff --git a/README.md b/README.md
index 3add72c..4802ed1 100644
--- a/README.md
+++ b/README.md
@@ -7,4 +7,4 @@ Subject Name: Project Management with Git
 Subject Code: BCS358C
 Academic Year:  2024-25
 Mobile No: 9620483405
-
+My Date of Joining to GITW: 3rd Oct-2023
```

Author name, Email, Date,timings and committed messages are shown.

Similarly for feature-branch:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log feature-branch --oneline
```

It will show:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log feature-branch --oneline
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
ac80e2e Removed Email ID
d9068ac Added README.md file successfully
4dda3ea Added README.md file with basic data successfully
```

Now for the given ID c3a5454use:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git show c3a5454
```

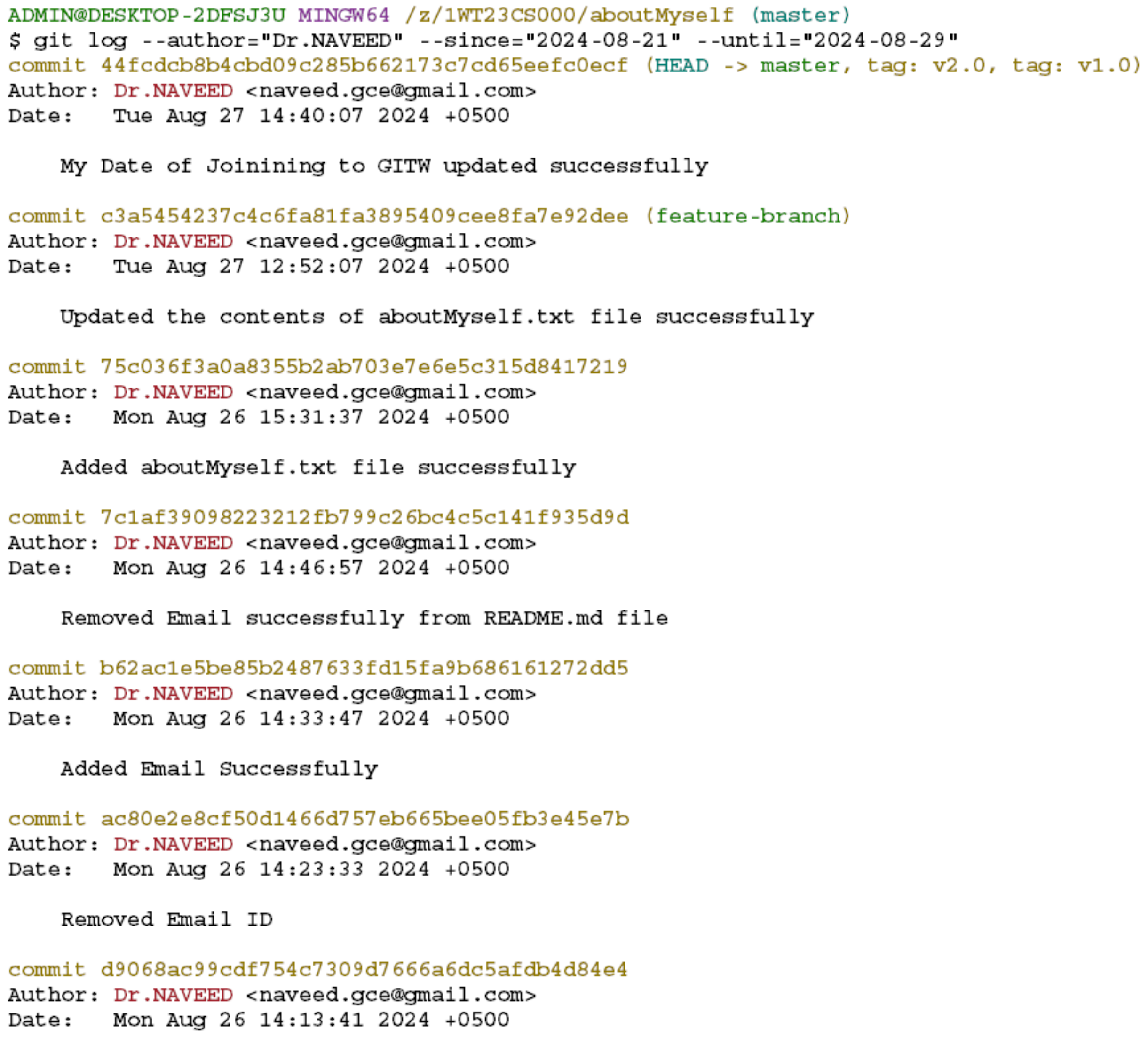
It will show.

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git show c3a5454
commit c3a5454237c4c6fa81fa3895409cee8fa7e92dee (feature-branch)
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Tue Aug 27 12:52:07 2024 +0500

    Updated the contents of aboutMyself.txt file successfully

diff --git a/aboutMyself.txt b/aboutMyself.txt
index d608607..daee66a 100644
--- a/aboutMyself.txt
+++ b/aboutMyself.txt
@@ -6,3 +6,6 @@ Section: A
 Branch : Mechanical Engineering
 Year of Admission: 2023
 College Name: GITW
+UG Degree: Mechanical Engineering
+PG Degree: Manufacturing Science & Engineering
+PhD Degree: Materials Science
```

To list all commits made by the author "Dr. NAVEED" between "2024-01-01" and "2024-12-31."

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log --author="Dr.NAVEED" --since="2024-08-21" --until="2024-08-29"
```

It should be "Year-Month-Day" format. It will show:

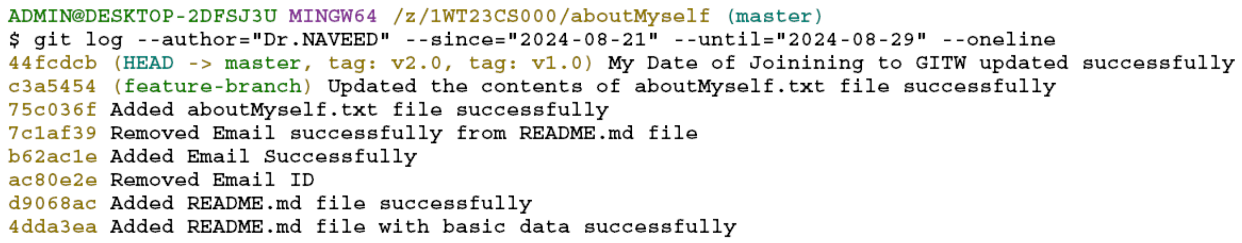


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log --author="Dr.NAVEED" --since="2024-08-21" --until="2024-08-29"
commit 44fcdcb8b4cbd09c285b662173c7cd65eefc0ecf (HEAD -> master, tag: v2.0, tag: v1.0)
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Tue Aug 27 14:40:07 2024 +0500

    My Date of Joinining to GITW updated successfully

commit c3a5454237c4c6fa81fa3895409cee8fa7e92dee (feature-branch)
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Tue Aug 27 12:52:07 2024 +0500

    Updated the contents of aboutMyself.txt file successfully

commit 75c036f3a0a8355b2ab703e7e6e5c315d8417219
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Mon Aug 26 15:31:37 2024 +0500

    Added aboutMyself.txt file successfully

commit 7c1af39098223212fb799c26bc4c5c141f935d9d
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Mon Aug 26 14:46:57 2024 +0500

    Removed Email successfully from README.md file

commit b62ac1e5be85b2487633fd15fa9b686161272dd5
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Mon Aug 26 14:33:47 2024 +0500

    Added Email Successfully

commit ac80e2e8cf50d1466d757eb665bee05fb3e45e7b
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Mon Aug 26 14:23:33 2024 +0500

    Removed Email ID

commit d9068ac99cdf754c7309d7666a6dc5afdb4d84e4
Author: Dr.NAVEED <naveed.gce@gmail.com>
Date:   Mon Aug 26 14:13:41 2024 +0500
```

Note: Press Q to get back to coding.
If we require the details in brief just in one line then use:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log --author="Dr.NAVEED" --since="2024-08-21" --until="2024-08-29" --oneline
```

It will show:

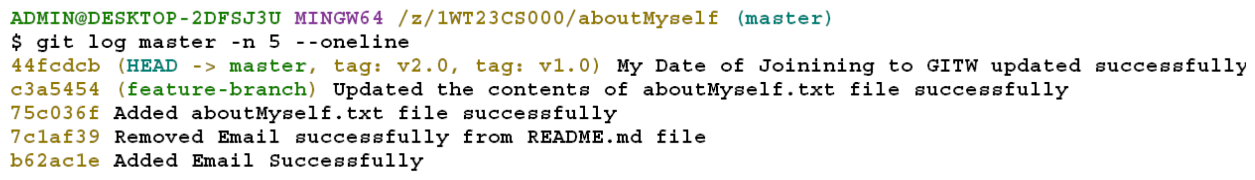


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log --author="Dr.NAVEED" --since="2024-08-21" --until="2024-08-29" --oneline
44fcdcb (HEAD -> master, tag: v2.0, tag: v1.0) My Date of Joinining to GITW updated successfully
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
ac80e2e Removed Email ID
d9068ac Added README.md file successfully
4dda3ea Added README.md file with basic data successfully
```

To display the last five commits in the repository's history for master branch:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 5 --oneline
```

It will show:



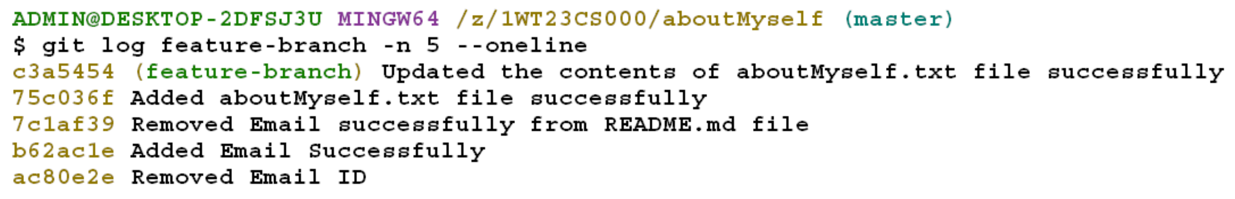
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 5 --oneline
44fcdcb (HEAD -> master, tag: v2.0, tag: v1.0) My Date of Joinining to GITW updated successfully
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
```

Note: To get full details avoid --oneline:

Similarly for feature-branch:

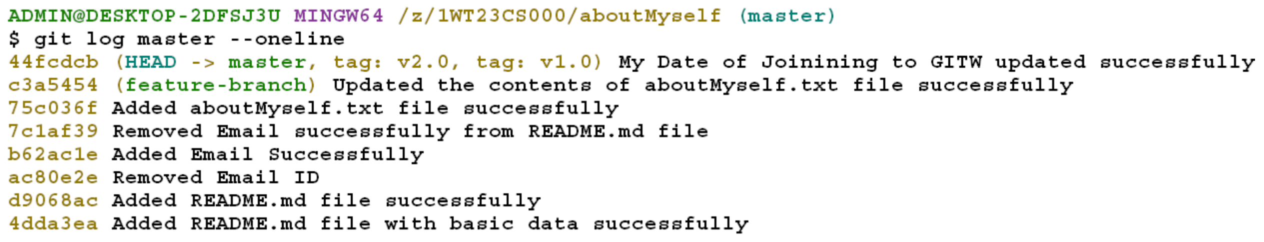
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log feature-branch -n 5 --oneline
```

It will show:

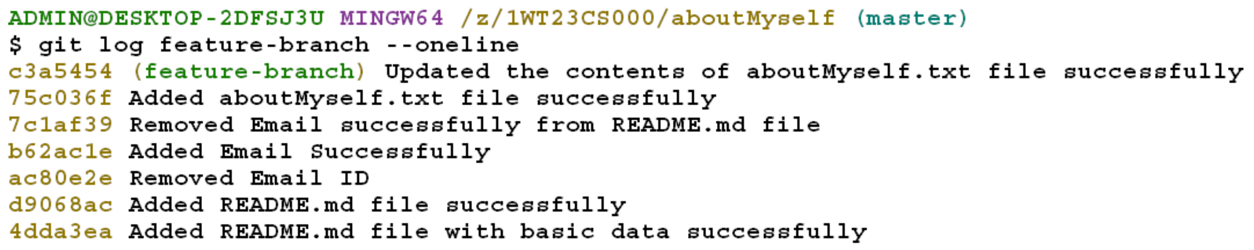


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log feature-branch -n 5 --oneline
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
ac80e2e Removed Email ID
```

OUTPUT:

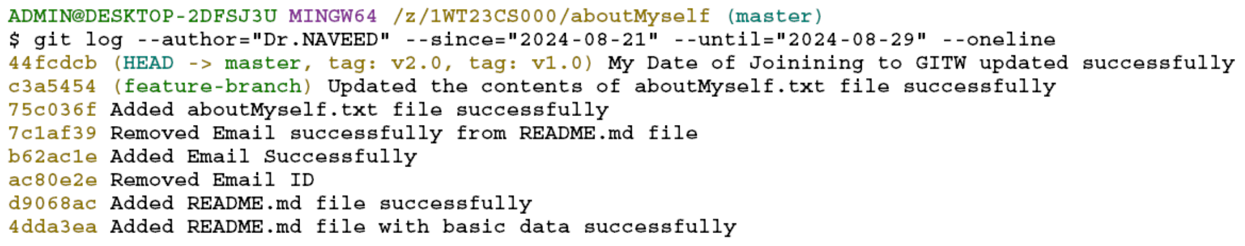


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master --oneline
44fcdcb (HEAD -> master, tag: v2.0, tag: v1.0) My Date of Joinining to GITW updated successfully
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
ac80e2e Removed Email ID
d9068ac Added README.md file successfully
4dda3ea Added README.md file with basic data successfully
```

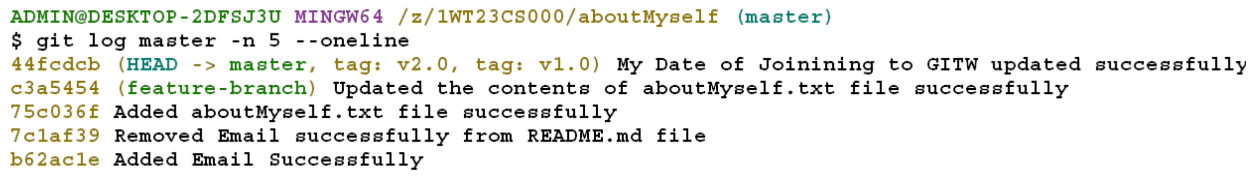


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log feature-branch --oneline
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
ac80e2e Removed Email ID
d9068ac Added README.md file successfully
4dda3ea Added README.md file with basic data successfully
```

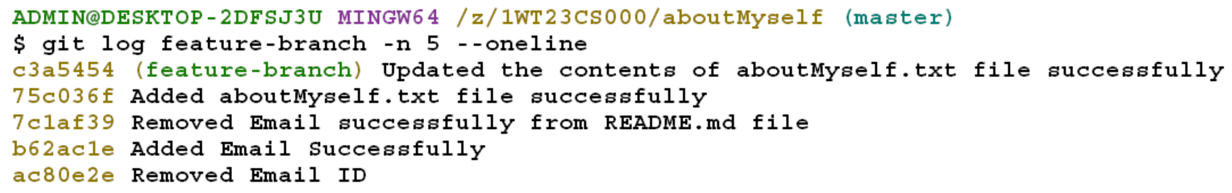
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log --author="Dr.NAVEED" --since="2024-08-21" --until="2024-08-29" --oneline
```



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log --author="Dr.NAVEED" --since="2024-08-21" --until="2024-08-29" --oneline
44fcdcb (HEAD -> master, tag: v2.0, tag: v1.0) My Date of Joinining to GITW updated successfully
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
ac80e2e Removed Email ID
d9068ac Added README.md file successfully
4dda3ea Added README.md file with basic data successfully
```



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 5 --oneline
44fcdcb (HEAD -> master, tag: v2.0, tag: v1.0) My Date of Joinining to GITW updated successfully
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
```



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log feature-branch -n 5 --oneline
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
ac80e2e Removed Email ID
```

Experiment-08: GIT Cherry-pick and Revert

Aim:

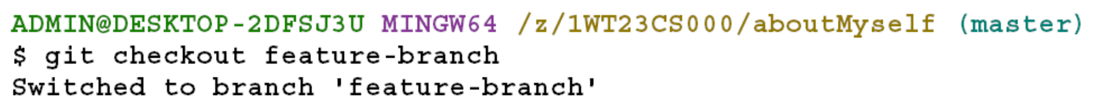
1. Create a new file “bugfile.txt” in feature-branch and obtain the details such as day and time of this file from master branch without merging it
2. To check committed message with date and time by using the reference ID in any branch such as feature-branch for the ID: c3a5454 by using cherry-pick command. Resolve the error if it exist.
3. To write the command to undo the changes introduced by the commit with the ID "b5b6caf" in master branch. Change the committed message from “Deleted data1.txt file successfully” to Deleted data1.txt file successfully which was a dummy file for testing purpose”. Resolve the error if it exist.

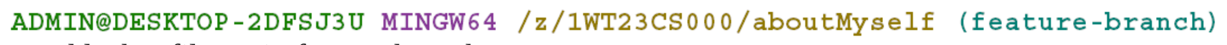
The git cherry-pick command is used to apply the changes introduced by an existing commit onto the current branch. It allows you to select a specific commit from one branch and apply it to another branch, without merging the entire branch history. For example, if you want to add some committed message done in feature-branch into the master branch without merging the feature-branch into the master branch then git cherry-pick is helpful. This can be useful in a variety of situations, such as:

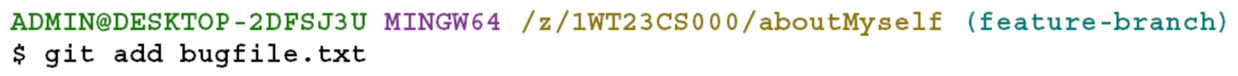
1. **Porting Specific Changes:** If you've made a specific fix or feature in one branch and want to apply it to another branch without merging all other changes from the source branch, you can use git cherry-pick to apply just that commit.
2. **Selective Backporting:** When you need to backport, a bug fixes from a development branch to a stable branch without including all other changes.
3. **Undoing Changes:** You can also use git cherry-pick in combination with a revert commit to undo specific changes made by a particular commit.

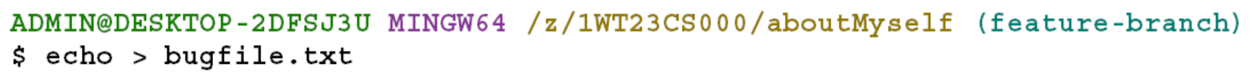
To create a new file “bugfile.txt”:

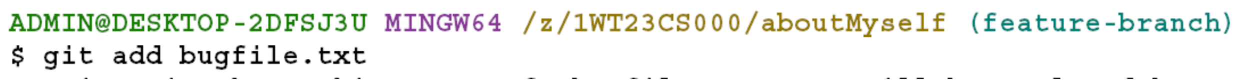
First, we will shift to feature-branch by using

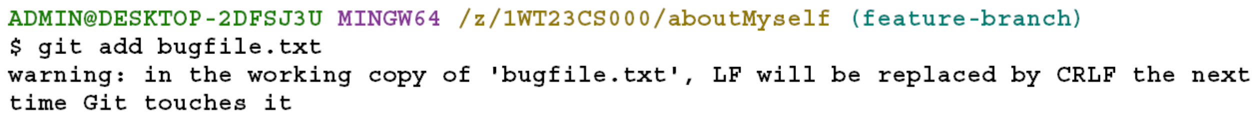

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git checkout feature-branch
Switched to branch 'feature-branch'
```

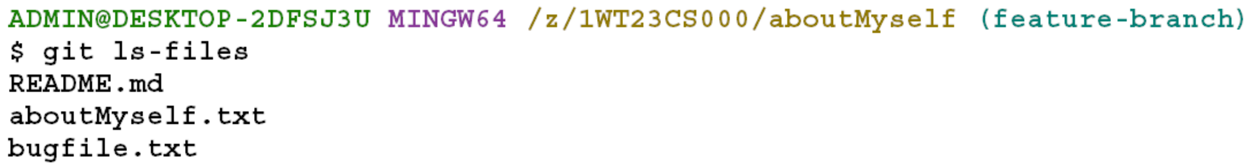

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
```
To add a bugfile.txt in feature-branch use:

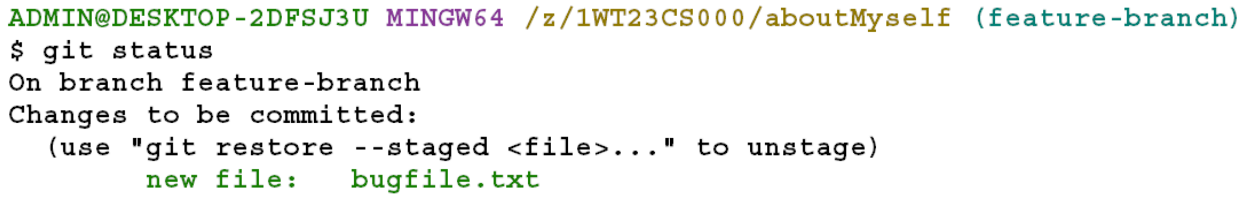

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git add bugfile.txt
```
It shows some error: We will use echo >bugfile.txt and then use git add

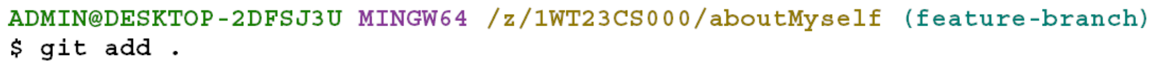

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ echo > bugfile.txt
```
It will add an untraced file bugfile.txt. To trace it and add it use:

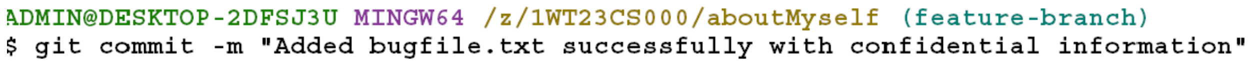

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git add bugfile.txt
```
It will show:

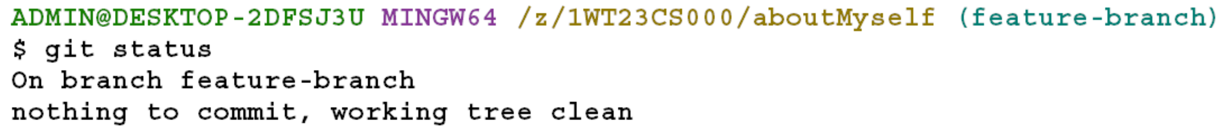

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git add bugfile.txt
warning: in the working copy of 'bugfile.txt', LF will be replaced by CRLF the next
time Git touches it
```
Ignore the warning as it is related to type of operating system.Now if we check the number of files.


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git ls-files
README.md
aboutMyself.txt
bugfile.txt
```
We can see bugfile.txt is added. If we check the status:

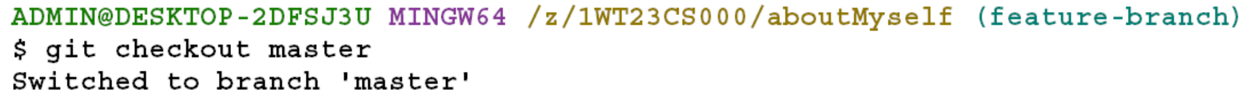

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git status
On branch feature-branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   bugfile.txt
```
It shows new file and it will ask to commit. To commit with a message "Added bugfile.txt successfully with confidential information" use:

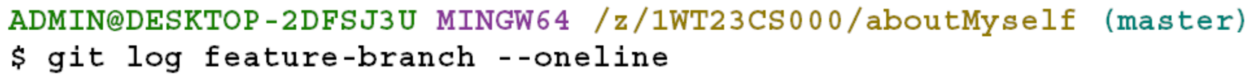

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git add .
```

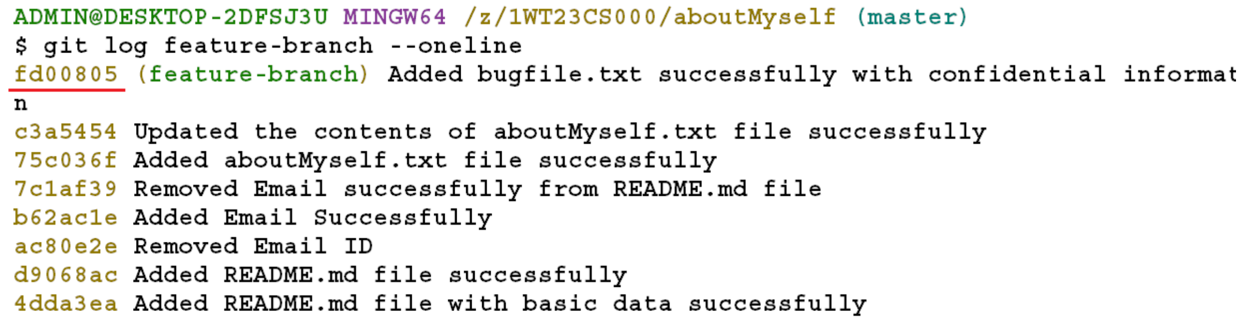

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git commit -m "Added bugfile.txt successfully with confidential information"
```
Now if we check the status

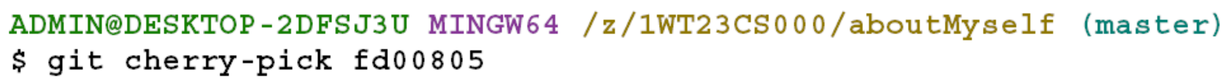

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git status
On branch feature-branch
nothing to commit, working tree clean
```

Everything is fine.Now to check this committed message in master branch without merging feature-branch into master branch, we will shift to master branch.


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git checkout master
Switched to branch 'master'
```
Let us find out the history of feature-branch to get the reference ID related to bugfile.txt by using:

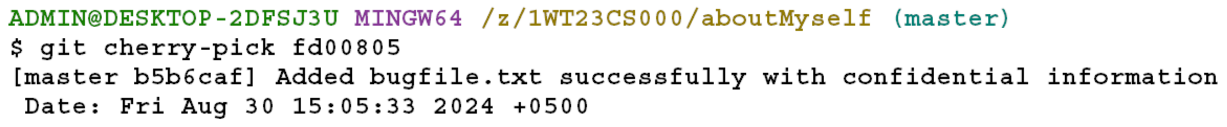

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log feature-branch --oneline
```
It will show:


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log feature-branch --oneline
fd00805 (feature-branch) Added bugfile.txt successfully with confidential informat
n
c3a5454 Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
ac80e2e Removed Email ID
d9068ac Added README.md file successfully
4dda3ea Added README.md file with basic data successfully
```
From the above we got the ID as: fd00805. By using



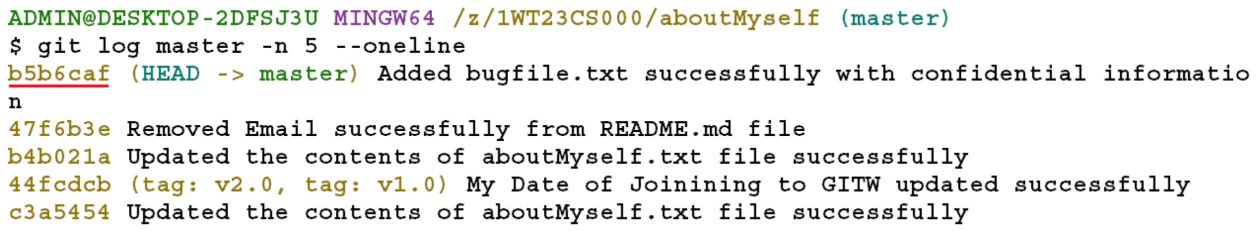
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git cherry-pick fd00805
```

We can get information about the time and date details of the bugfile.txt inserted into the feature-branch. It shows



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git cherry-pick fd00805
[master b5b6caf] Added bugfile.txt successfully with confidential information
 Date: Fri Aug 30 15:05:33 2024 +0500
```

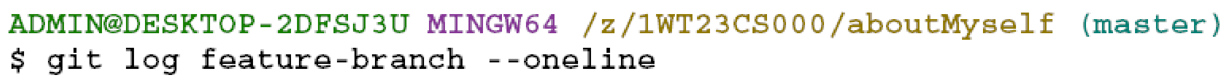
Now if we check the history of master branch



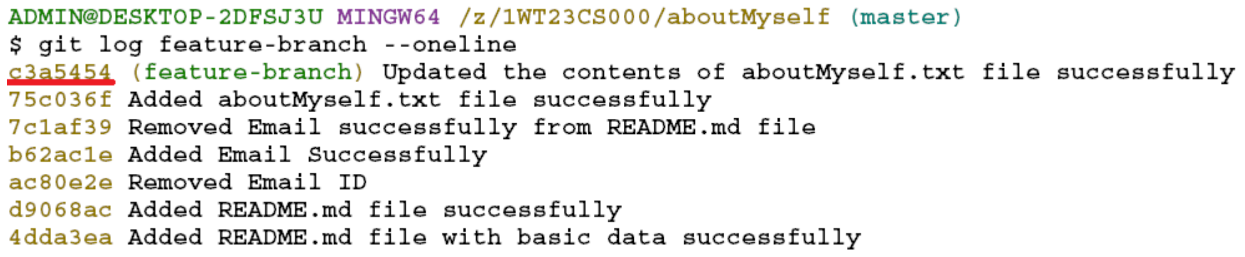
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 5 --oneline
b5b6caf (HEAD -> master) Added bugfile.txt successfully with confidential informatio
n
47f6b3e Removed Email successfully from README.md file
b4b021a Updated the contents of aboutMyself.txt file successfully
44fcdcb (tag: v2.0, tag: v1.0) My Date of Joinining to GITW updated successfully
c3a5454 Updated the contents of aboutMyself.txt file successfully
```

We can see that the committed message added in feature-branch related to bugfile.txt is now saved in master branch.
Therefore, date and time of the bugfile.txt is obtained successfully without merging feature-branch into the master branch by using git cherry-pick.

To check committed message with date and time by using the reference ID in any branch such as feature-branch in the above ID c3a5454by cherry-pick command:
Use git cherry-pick command for thegiven IDin feature-branch: as shown below
First, we will use:

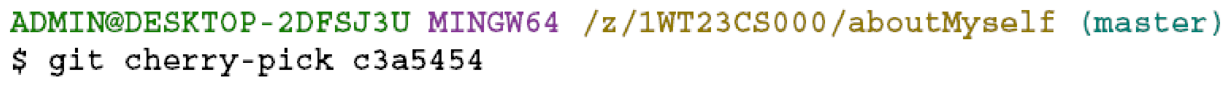


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log feature-branch --oneline
```



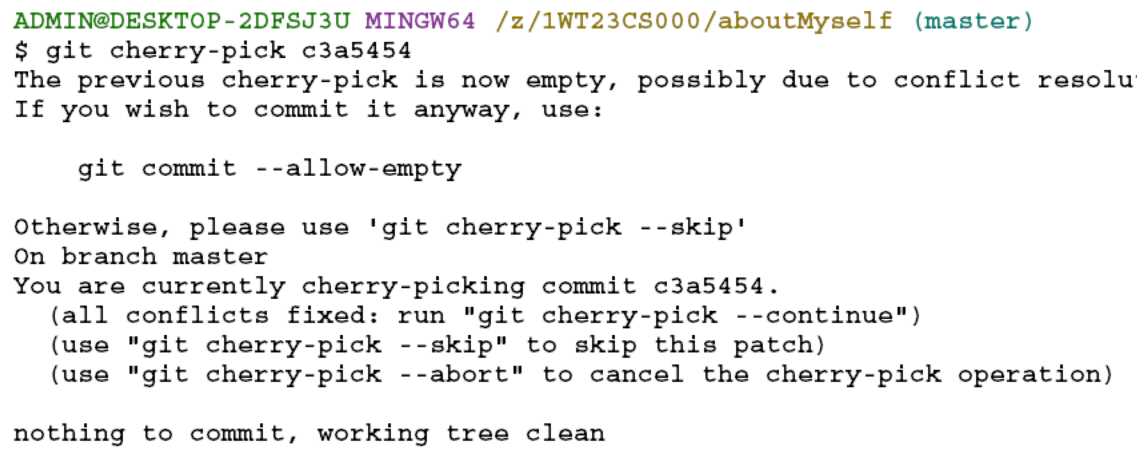
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log feature-branch --oneline
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
ac80e2e Removed Email ID
d9068ac Added README.md file successfully
4dda3ea Added README.md file with basic data successfully
```

To check the details of the committed message with date and time for the above ID use:



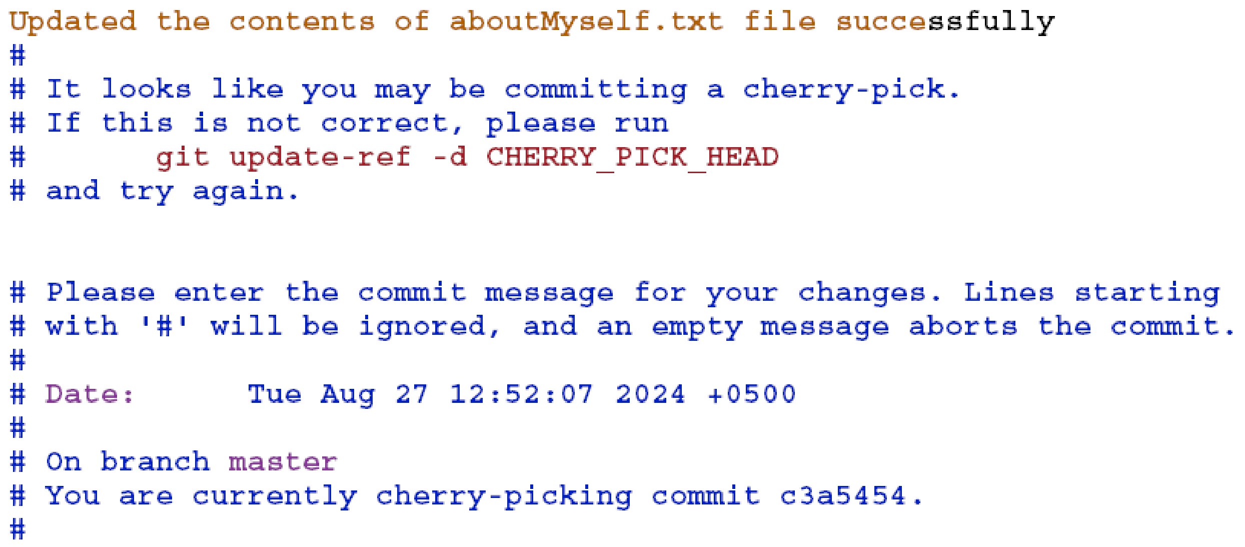
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git cherry-pick c3a5454
```

It will show the details if no error found. If error found it will show the error.



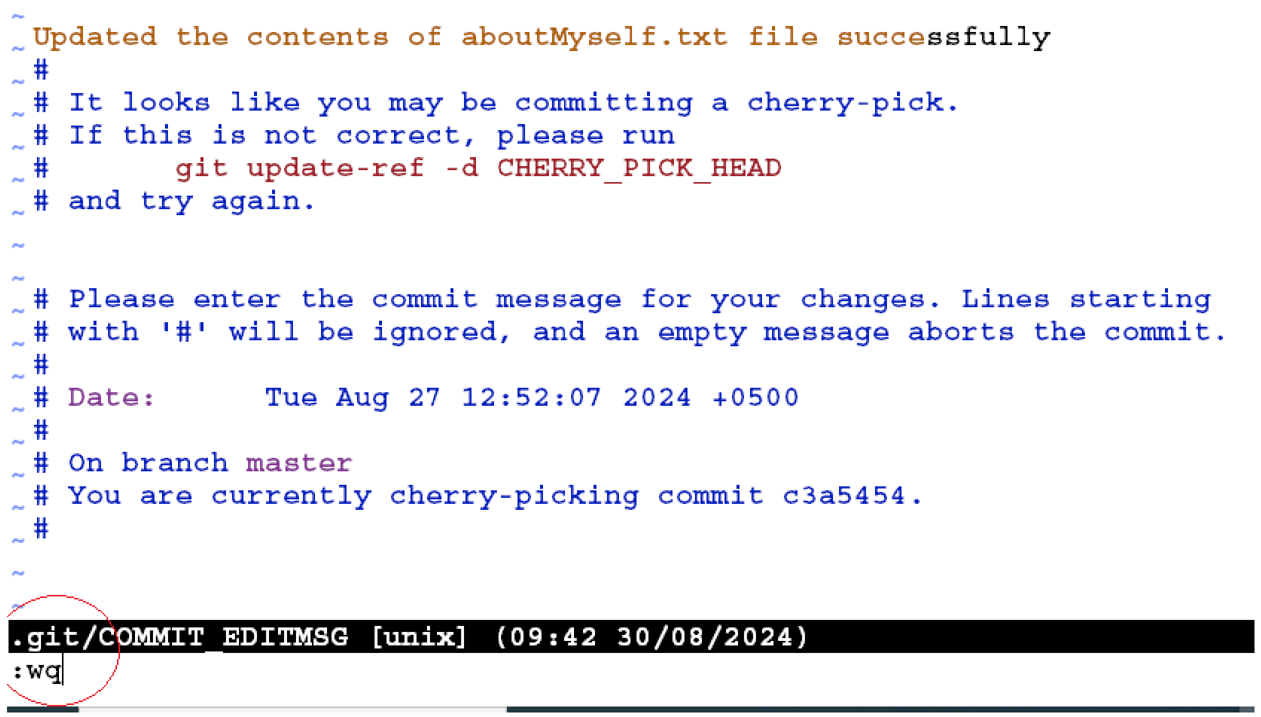
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git cherry-pick c3a5454
The previous cherry-pick is now empty, possibly due to conflict resolution.
If you wish to commit it anyway, use:

    git commit --allow-empty

Otherwise, please use 'git cherry-pick --skip'
On branch master
You are currently cherry-picking commit c3a5454.
  (all conflicts fixed: run "git cherry-pick --continue")
  (use "git cherry-pick --skip" to skip this patch)
  (use "git cherry-pick --abort" to cancel the cherry-pick operation)

nothing to commit, working tree clean
```

It tells that the cherry-pick is now empty due to conflict resolution. Follow the instruction given above. We will use git commit --allow-empty. A window will open as shown below:



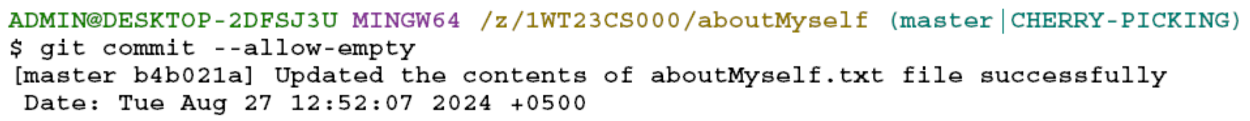
```
Updated the contents of aboutMyself.txt file successfully
#
# It looks like you may be committing a cherry-pick.
# If this is not correct, please run
#       git update-ref -d CHERRY_PICK_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Tue Aug 27 12:52:07 2024 +0500
#
# On branch master
# You are currently cherry-picking commit c3a5454.
#
```

The committed message is shown in brown color. If you want to modify the message it can be modified or else keep as such.



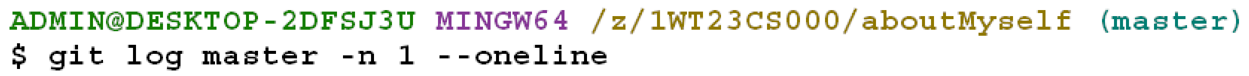
```
~
~Updated the contents of aboutMyself.txt file successfully
~#
~# It looks like you may be committing a cherry-pick.
~# If this is not correct, please run
~#       git update-ref -d CHERRY_PICK_HEAD
~# and try again.
~
~
~
~# Please enter the commit message for your changes. Lines starting
~# with '#' will be ignored, and an empty message aborts the commit.
~#
~# Date:      Tue Aug 27 12:52:07 2024 +0500
~#
~# On branch master
~# You are currently cherry-picking commit c3a5454.
~#
~
~
.git/COMMIT_EDITMSG [unix]  (09:42 30/08/2024)
:wq
```

To get back to the coding screen **press Esc button** and then type ":wq" at the footer as shown above and then enter.
Now it will show the details of the commit.

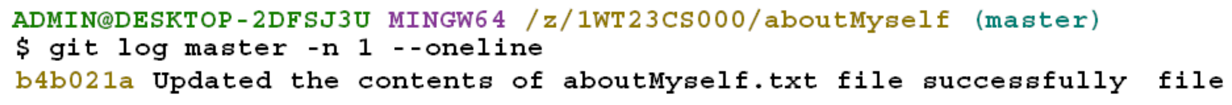


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master|CHERRY-PICKING)
$ git commit --allow-empty
[master b4b021a] Updated the contents of aboutMyself.txt file successfully
 Date: Tue Aug 27 12:52:07 2024 +0500
```

We can see that the above committed message of feature-branch with the ID: c3a5454 is now saved in master branch.
To check use:



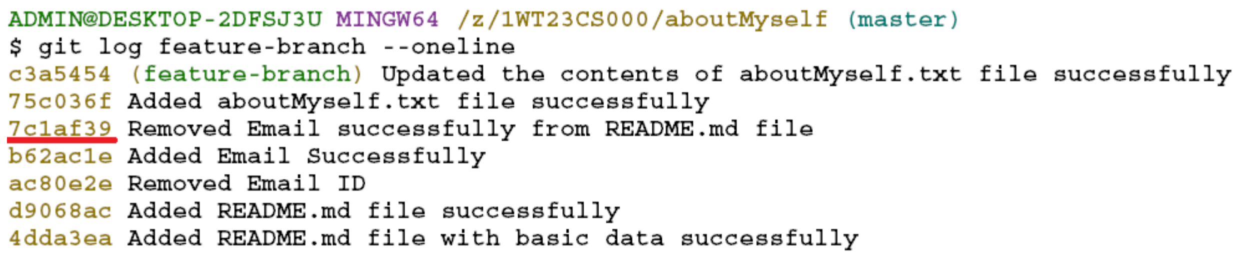
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 1 --oneline
```



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 1 --oneline
b4b021a Updated the contents of aboutMyself.txt file successfully  file
```

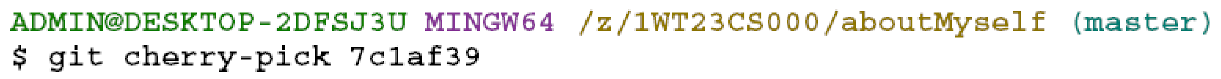
We can see that the committed message of feature-branch is now added into the master branch with a new ID b4b021a. This is helpful if we want to share some important commit of feature-branch into the master branch without merging the entire feature-branch into the master branch.

Let us check some other ID such as 7c1af39 as shown below.



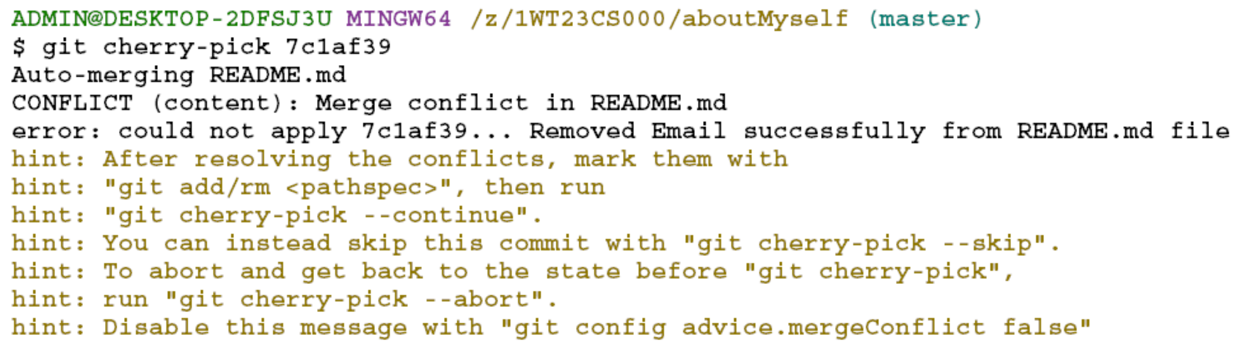
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log feature-branch --oneline
c3a5454 (feature-branch) Updated the contents of aboutMyself.txt file successfully
75c036f Added aboutMyself.txt file successfully
7c1af39 Removed Email successfully from README.md file
b62ac1e Added Email Successfully
ac80e2e Removed Email ID
d9068ac Added README.md file successfully
4dda3ea Added README.md file with basic data successfully
```

Use the git cherry-pick command for the above ID.



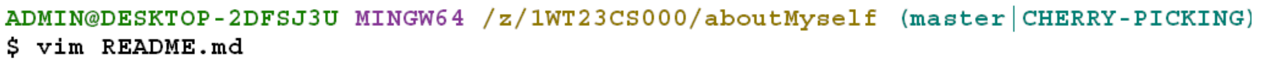
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git cherry-pick 7c1af39
```

It shows some error.



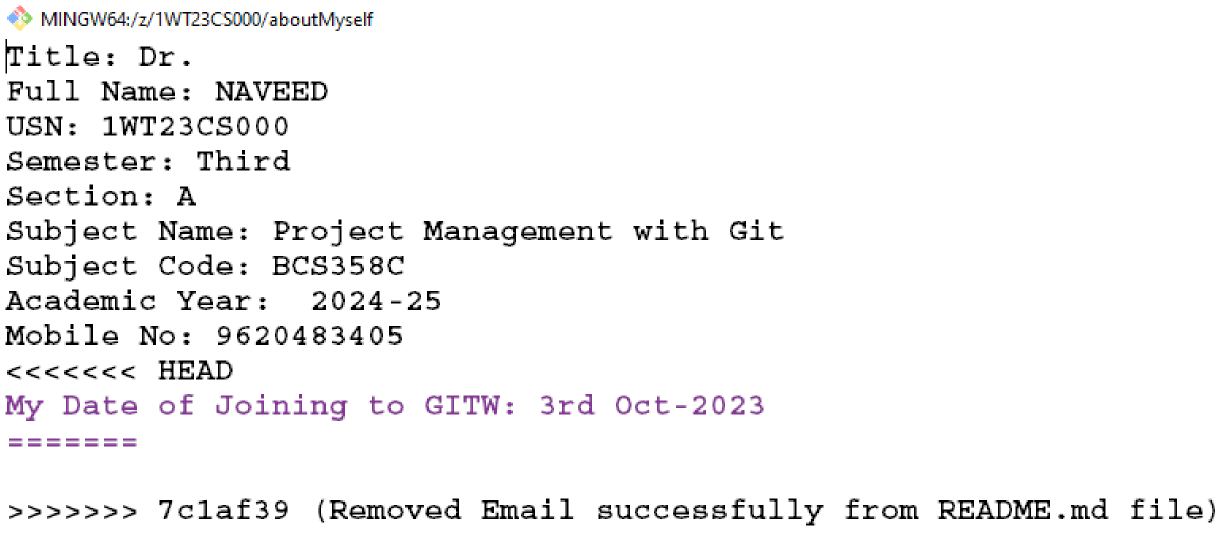
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git cherry-pick 7c1af39
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
error: could not apply 7c1af39... Removed Email successfully from README.md file
hint: After resolving the conflicts, mark them with
hint: "git add/rm <pathspec>", then run
hint: "git cherry-pick --continue".
hint: You can instead skip this commit with "git cherry-pick --skip".
hint: To abort and get back to the state before "git cherry-pick",
hint: run "git cherry-pick --abort".
hint: Disable this message with "git config advice.mergeConflict false"
```

To open the file use:



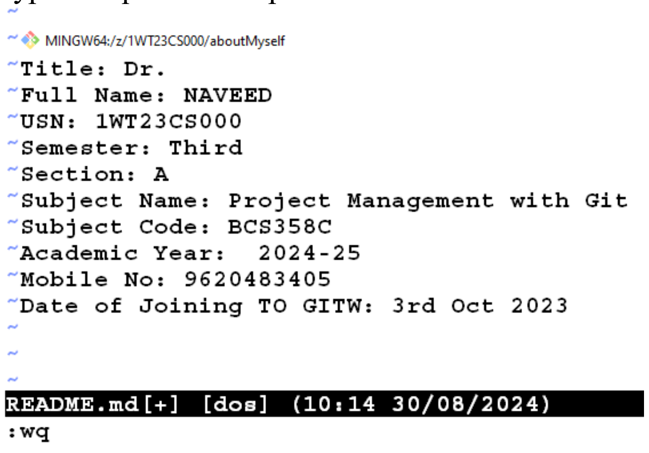
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master|CHERRY-PICKING)
$ vim README.md
```

It will open the file README.md.



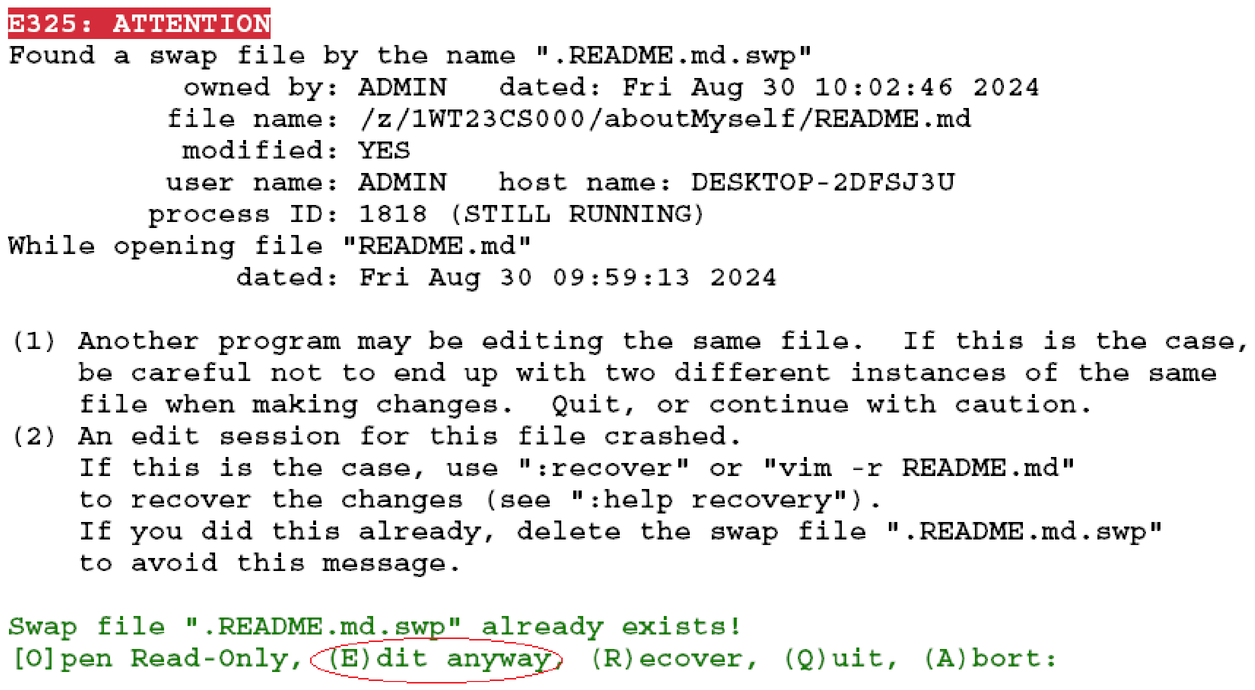
```
MINGW64:/z/1WT23CS000/aboutMyself
Title: Dr.
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Subject Name: Project Management with Git
Subject Code: BCS358C
Academic Year:  2024-25
Mobile No: 9620483405
<<<<<<< HEAD
My Date of Joining to GITW: 3rd Oct-2023
=======

>>>>>>> 7c1af39 (Removed Email successfully from README.md file)
```

Remove the conflicting text and edit the content as shown below. Keep the cursor at the text and type ":wq" and then press enter.



```
~
~ MINGW64:/z/1WT23CS000/aboutMyself
~Title: Dr.
~Full Name: NAVEED
~USN: 1WT23CS000
~Semester: Third
~Section: A
~Subject Name: Project Management with Git
~Subject Code: BCS358C
~Academic Year:  2024-25
~Mobile No: 9620483405
~Date of Joining TO GITW: 3rd Oct 2023
~
~
~
README.md[+] [dos] (10:14 30/08/2024)
:wq
```

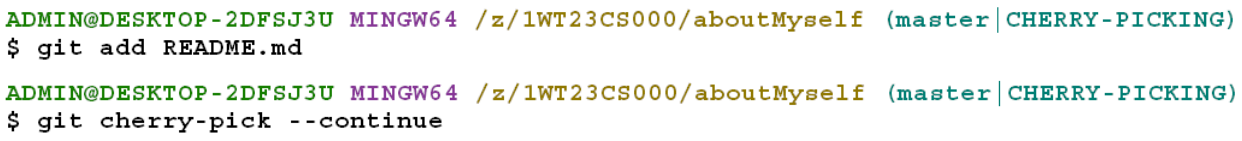
It will open an Attention file:



```
E325: ATTENTION
Found a swap file by the name ".README.md.swp"
          owned by: ADMIN   dated: Fri Aug 30 10:02:46 2024
         file name: /z/1WT23CS000/aboutMyself/README.md
          modified: YES
         user name: ADMIN   host name: DESKTOP-2DFSJ3U
        process ID: 1818 (STILL RUNNING)
While opening file "README.md"
             dated: Fri Aug 30 09:59:13 2024

(1) Another program may be editing the same file.  If this is the case,
    be careful not to end up with two different instances of the same
    file when making changes.  Quit, or continue with caution.
(2) An edit session for this file crashed.
    If this is the case, use ":recover" or "vim -r README.md"
    to recover the changes (see ":help recovery").
    If you did this already, delete the swap file ".README.md.swp"
    to avoid this message.

Swap file ".README.md.swp" already exists!
[O]pen Read-Only, [✓ (E)dit anyway,] (R)ecover, (Q)uit, (A)bort:
```

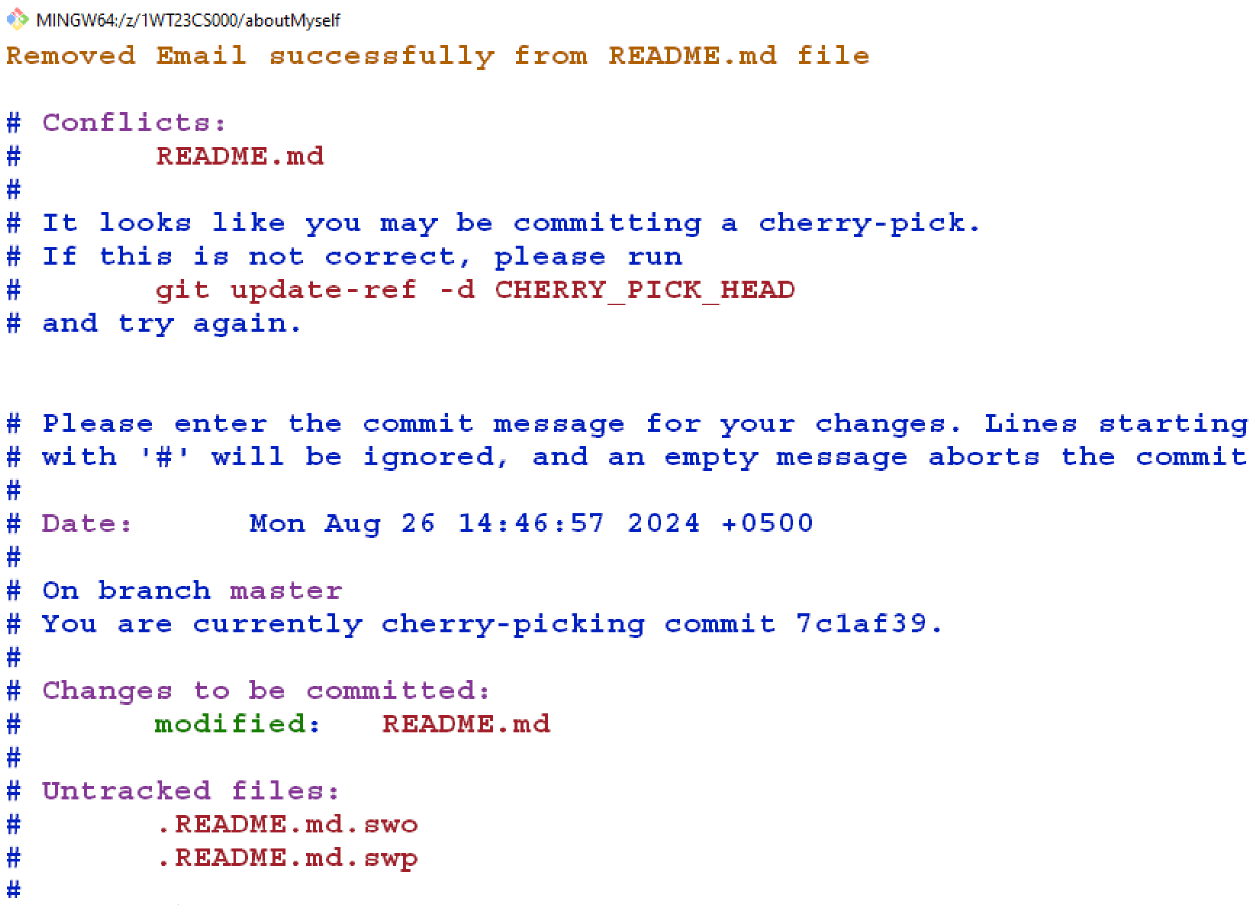
Type E and the proceed.

Add the modified file and continue with the cherry-pick



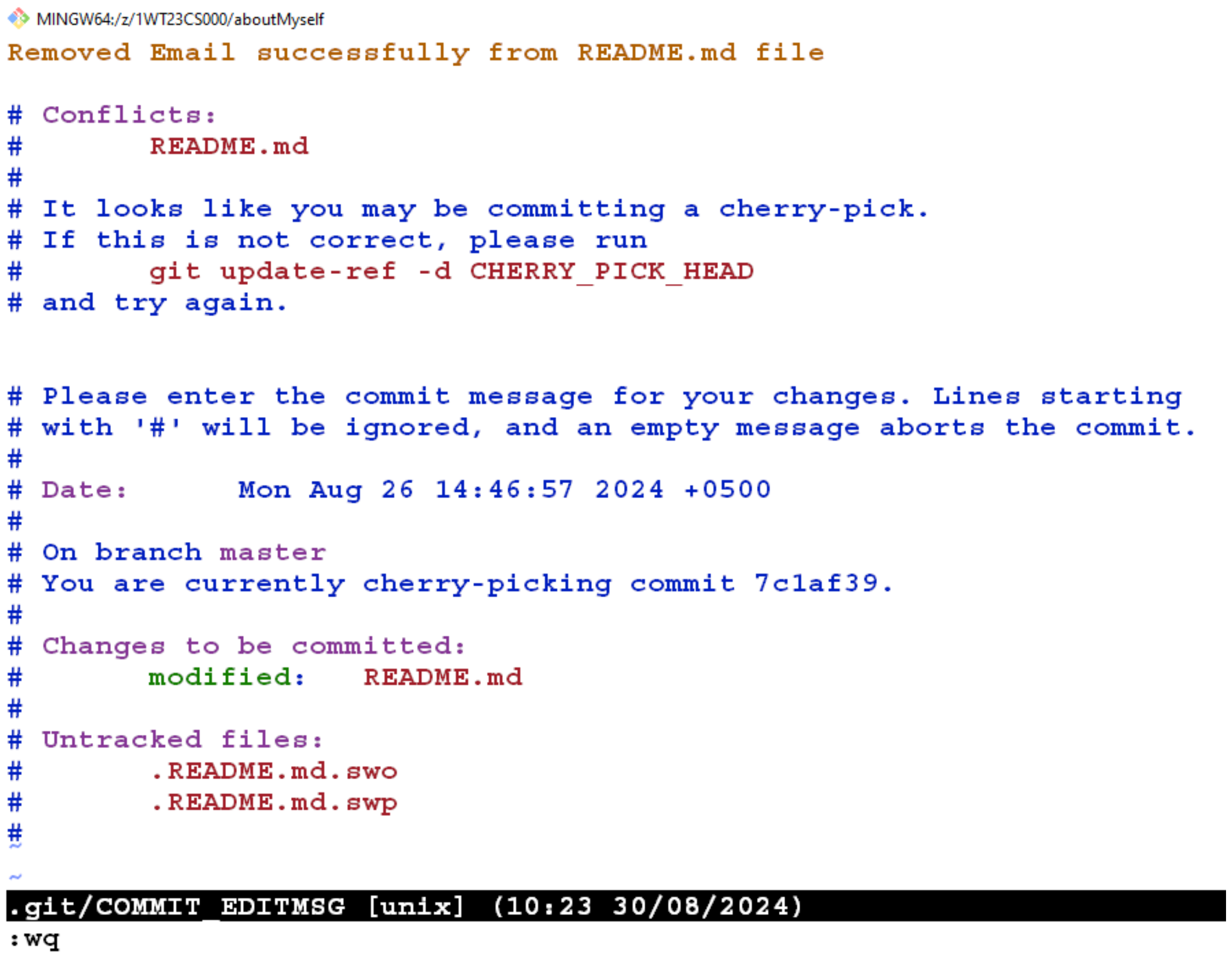
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master|CHERRY-PICKING)
$ git add README.md

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master|CHERRY-PICKING)
$ git cherry-pick --continue
```

It will open the conflicting file



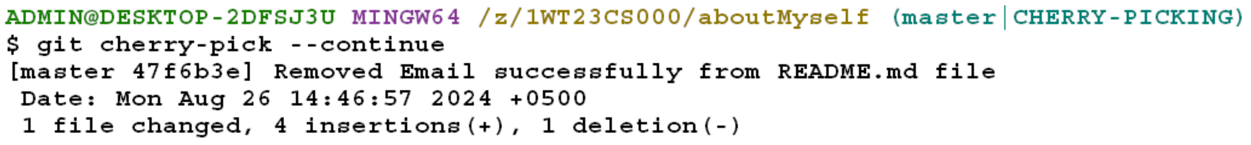
```
MINGW64:/z/1WT23CS000/aboutMyself
Removed Email successfully from README.md file

# Conflicts:
#       README.md
#
# It looks like you may be committing a cherry-pick.
# If this is not correct, please run
#       git update-ref -d CHERRY_PICK_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit
#
# Date:      Mon Aug 26 14:46:57 2024 +0500
#
# On branch master
# You are currently cherry-picking commit 7c1af39.
#
# Changes to be committed:
#       modified:   README.md
#
# Untracked files:
#       .README.md.swo
#       .README.md.swp
#
```

Type ":wq" then enter



```
Removed Email successfully from README.md file

# Conflicts:
#       README.md
#
# It looks like you may be committing a cherry-pick.
# If this is not correct, please run
#       git update-ref -d CHERRY_PICK_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:       Mon Aug 26 14:46:57 2024 +0500
#
# On branch master
# You are currently cherry-picking commit 7c1af39.
#
# Changes to be committed:
#       modified:   README.md
#
# Untracked files:
#       .README.md.swo
#       .README.md.swp
#
~
~
.git/COMMIT_EDITMSG [unix]  (10:23 30/08/2024)
:wq
```

Now we got the committed message.

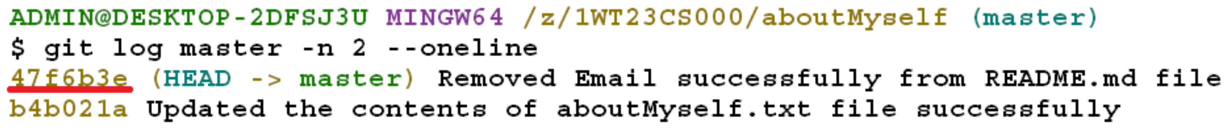


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master|CHERRY-PICKING)
$ git cherry-pick --continue
[master 47f6b3e] Removed Email successfully from README.md file
 Date: Mon Aug 26 14:46:57 2024 +0500
 1 file changed, 4 insertions(+), 1 deletion(-)
```

Now if we check the history of master branch for the last two commits

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 2 --oneline
```

It will show:



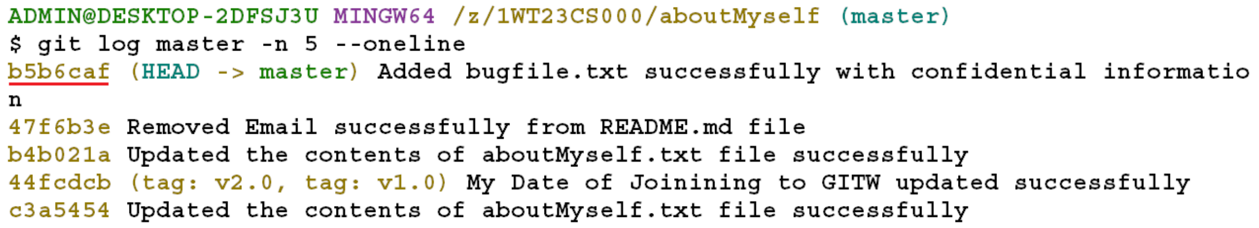
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 2 --oneline
47f6b3e (HEAD -> master) Removed Email successfully from README.md file
b4b021a Updated the contents of aboutMyself.txt file successfully
```

We can see that the committed message of feature-branch is now added into the master branch with a separate ID: 47f6b3e.

To undo the changes introduced by the commit with the ID "b5b6caf" in master branch.
First let us find out last 5 commits made in master branch by using:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 5 --oneline
```

It will show:

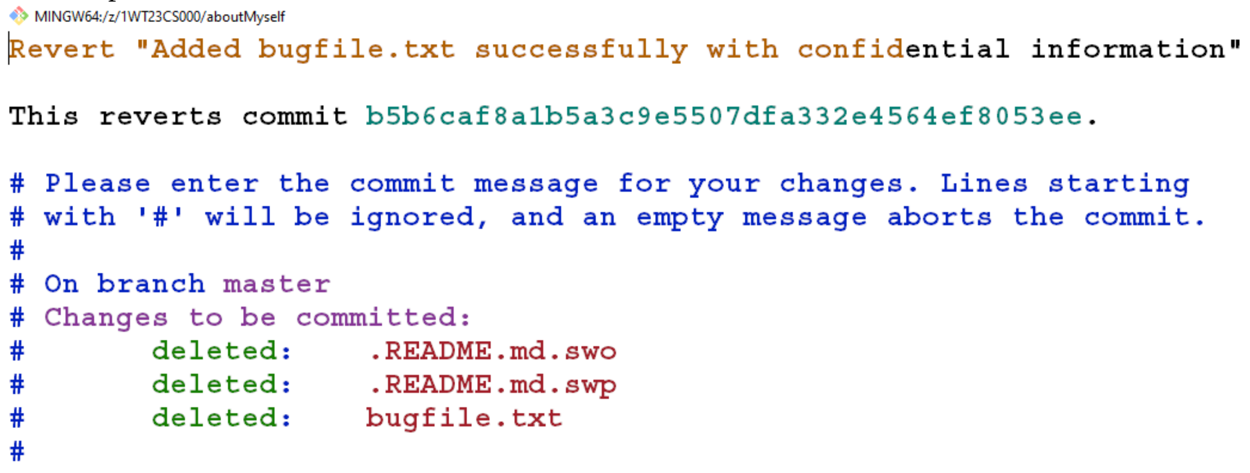


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 5 --oneline
b5b6caf (HEAD -> master) Added bugfile.txt successfully with confidential informatio
n
47f6b3e Removed Email successfully from README.md file
b4b021a Updated the contents of aboutMyself.txt file successfully
44fcdcb (tag: v2.0, tag: v1.0) My Date of Joinining to GITW updated successfully
c3a5454 Updated the contents of aboutMyself.txt file successfully
```

Now to change the committed message for the refence ID b5b6caf to "Added bugfile.txt successfully with very important information" use

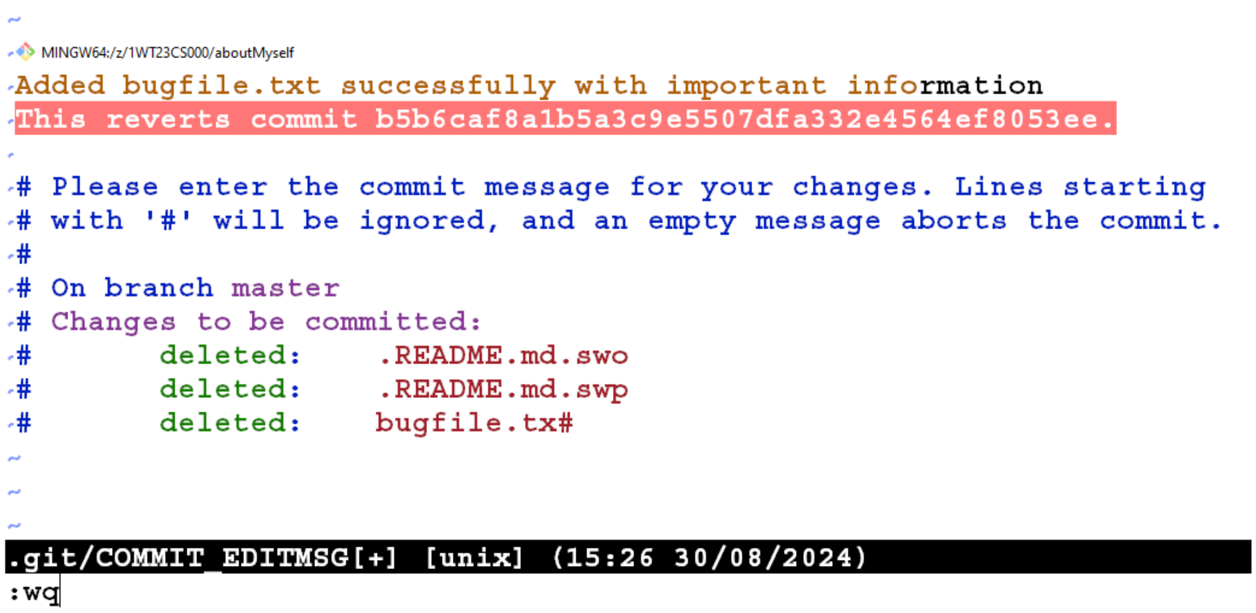
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git revert b5b6caf
```

It will open the file as shown below.



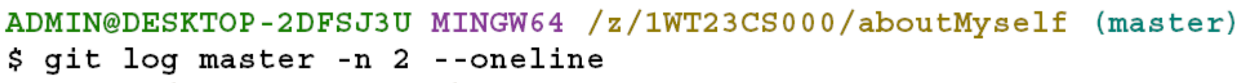
```
MINGW64:/z/1WT23CS000/aboutMyself
Revert "Added bugfile.txt successfully with confidential information"

This reverts commit b5b6caf8a1b5a3c9e5507dfa332e4564ef8053ee.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       deleted:    .README.md.swo
#       deleted:    .README.md.swp
#       deleted:    bugfile.txt
#
```

Edit the above file and change the content as to save and quiet type ":wq" as shown below:



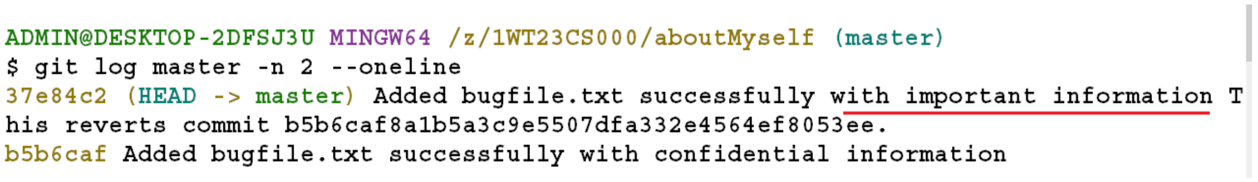
```
~
MINGW64:/z/1WT23CS000/aboutMyself
Added bugfile.txt successfully with important information
This reverts commit b5b6caf8a1b5a3c9e5507dfa332e4564ef8053ee.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       deleted:    .README.md.swo
#       deleted:    .README.md.swp
#       deleted:    bugfile.tx#
~
~
~
.git/COMMIT_EDITMSG[+] [unix] (15:26 30/08/2024)
:wq
```

Now if we check the last two committed message in master branch by using:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 2 --oneline
```

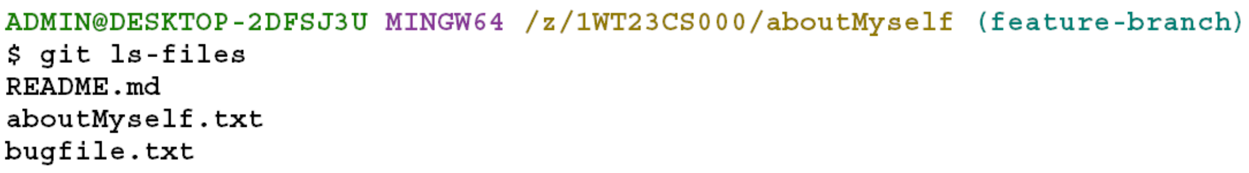
It will show :



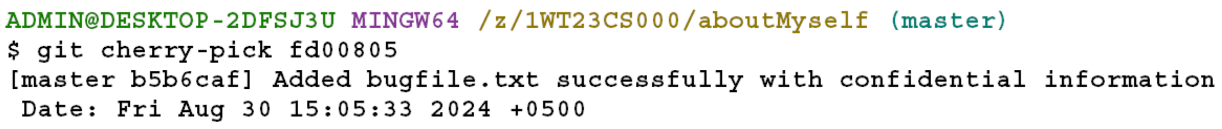
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 2 --oneline
37e84c2 (HEAD -> master) Added bugfile.txt successfully with important information T
his reverts commit b5b6caf8a1b5a3c9e5507dfa332e4564ef8053ee.
b5b6caf Added bugfile.txt successfully with confidential information
```

We can see the change of committed message in the above.

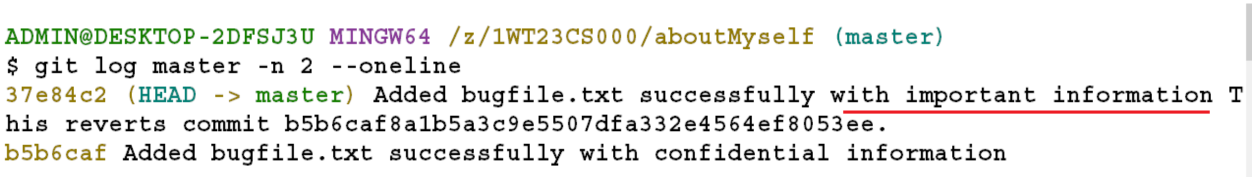
OUTPUT:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git ls-files
README.md
aboutMyself.txt
bugfile.txt
```



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git cherry-pick fd00805
[master b5b6caf] Added bugfile.txt successfully with confidential information
 Date: Fri Aug 30 15:05:33 2024 +0500
```



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git log master -n 2 --oneline
37e84c2 (HEAD -> master) Added bugfile.txt successfully with important information T
his reverts commit b5b6caf8a1b5a3c9e5507dfa332e4564ef8053ee.
b5b6caf Added bugfile.txt successfully with confidential information
```

GIT and VS Coding

Visual Studio Code (commonly referred to as VS Code) is a free, open-source code editor developed by Microsoft. It is designed to be lightweight yet powerful, with a rich set of features for developers. Here are some key aspects of VS Code. It is a versatile and powerful code editor that balances a lightweight footprint with rich functionality, making it a popular choice among developers of all kinds.

Key Features of VS Code:

1. Cross-Platform: Available on Windows, macOS, and Linux, making it accessible to a wide range of developers.

2.Extensible: It has a vast ecosystem of extensions available through the Visual Studio Code Marketplace. These extensions can enhance the editor with additional functionality, such as language support, linters, debuggers, and more.

3. Integrated Git Support: VS Code provides built-in Git integration, allowing users to perform Git operations like commits, branches, merges, and more directly from the editor.

4. Debugging: It has an integrated debugging tool that supports multiple programming languages and can be extended through plugins.

5. IntelliSense: Offers smart code completions based on variable types, function definitions, and imported modules, making it easier and faster to write code.

6.Customizable: Users can customize the editor's appearance and functionality through settings, themes, and keybindings.

7.Terminal Integration: Includes an integrated terminal that supports various shells (e.g., Bash, PowerShell), allowing developers to execute command-line operations within the editor.

8. Multi-Language Support: VS Code supports a wide range of programming languages out of the box and can be extended to support more through extensions.

9. Code Navigation and Refactoring: Provides features for easy navigation through code, such as "Go to Definition," "Peek Definition," and powerful refactoring tools.

10. Snippets and Emmet: Offers code snippets and Emmet support for faster coding.

Use Cases:

1.Software Development: Suitable for developing applications in various languages like JavaScript, Python, Java, C++, and more.
2.Web Development: Popular among web developers for working with HTML, CSS, JavaScript, and frameworks like React, Angular, and Vue.
3.Data Science: Can be used for data science tasks with extensions for Jupiter Notebooks, Python, and data visualization tools.
4.DevOps and Cloud: Integrated terminal and extensions for Docker, Kubernetes, and Azure make it useful for DevOps tasks.

5. Community and Ecosystem: VS Code has a large and active community contributing to its extensions and core features. The Visual Studio Code Marketplace offers a wide range of extensions that can significantly enhance the editor's functionality.

To download VS Code: Click the link given below:

https://drive.google.com/file/d/1q1zzTLY_ELwFg2mS2wJZRn3gD1V0CRQ7/view?usp=drive_link

Experiment-09: **Git in VS Code**

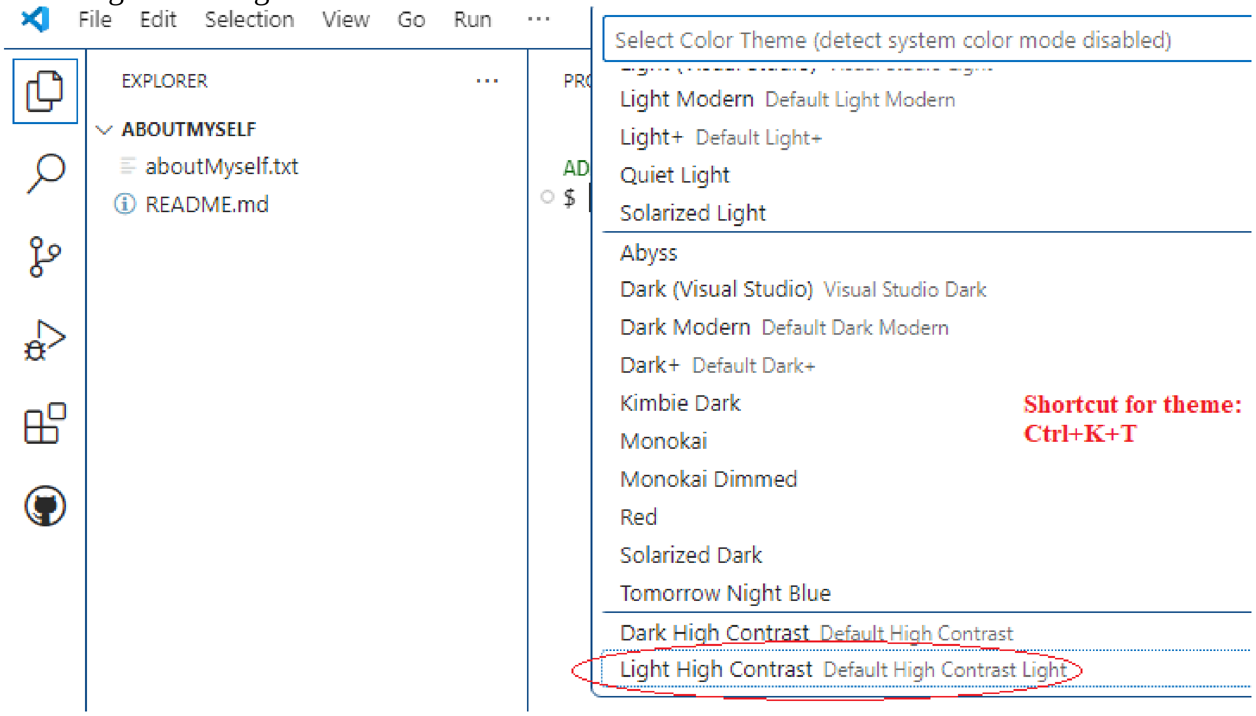
Aim:

1. To clone the repository /z/1WT23CS000/aboutMyself from Git-Bash to VS code:
2. To add a new file vsFile.txt under VS code. Add the following data:
   Vs File details:
   - Version: 2.1
   - Date of Installation: 30/07/2024
   - Author Name: Dr.Naveed

   and commit a message "Added vsFile.txt successfully"

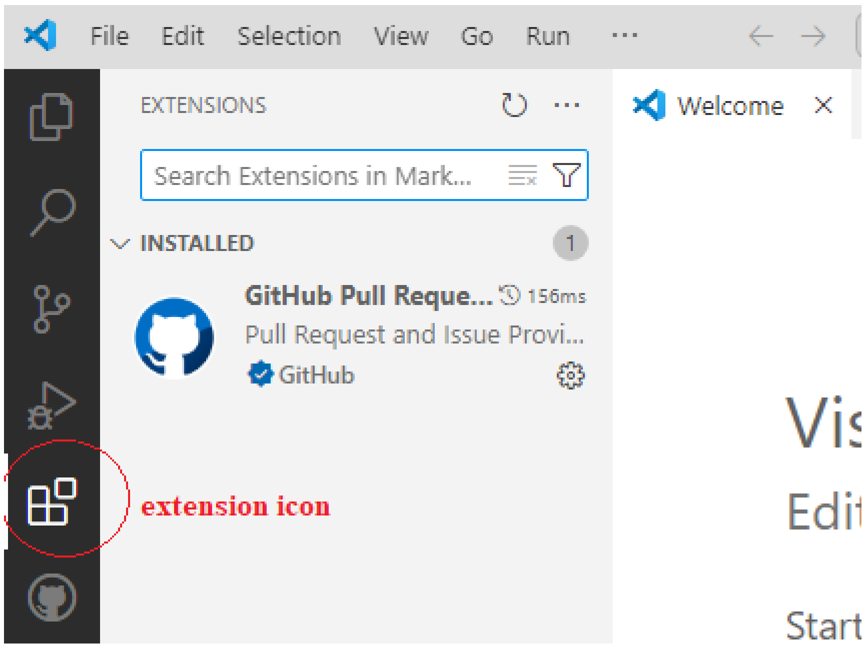
Basic Setup after Installation:

To change the background Theme to white use Ctrl+K+T


File
Edit
Selection
View
Go
Run
EXPLORER
ABOUTMYSELF
aboutMyself.txt
README.md
Select Color Theme (detect system color mode disabled)
Light Modern Default Light Modern
Light+ Default Light+
Quiet Light
Solarized Light
Abyss
Dark (Visual Studio) Visual Studio Dark
Dark Modern Default Dark Modern
Dark+ Default Dark+
Kimbie Dark
Monokai
Monokai Dimmed
Red
Solarized Dark
Tomorrow Night Blue
Dark High Contrast Default High Contrast
Light High Contrast Default High Contrast Light
Shortcut for theme: Ctrl+K+T

Select Light High Contrast as shown above.

Under Extension search for GitHub and Install it.


File
Edit
Selection
View
Go
Run
EXTENSIONS
Welcome
Search Extensions in Mark...
INSTALLED
1
GitHub Pull Reque... 156ms
Pull Request and Issue Provi...
GitHub
extension icon
Vis
Edi
Start

To clone the repository from git bash to VS code platform:
First, we will shift to the required repository position by using:

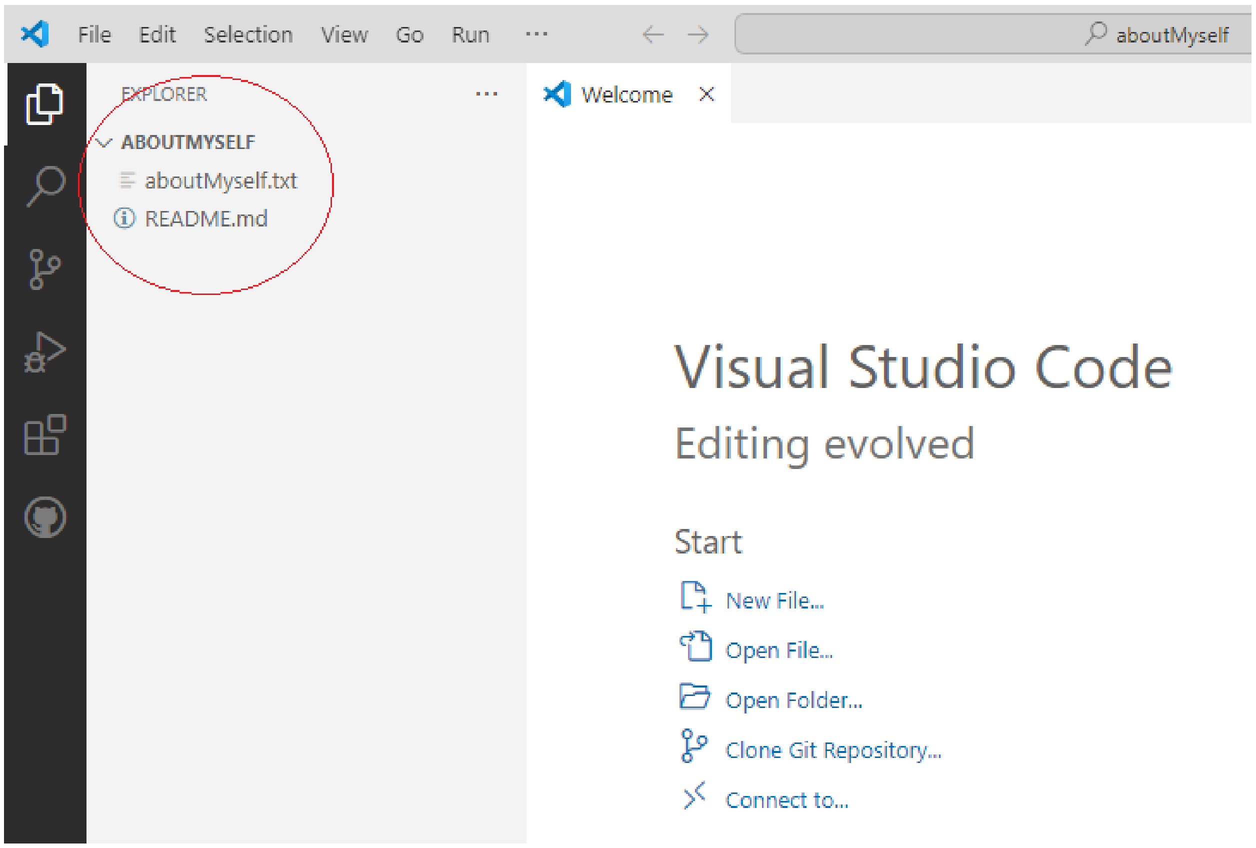
MINGW64:/Z/1WT23CS000/aboutMyself

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 ~
$ cd /Z

ADMIN@DESKTOP-2DFSJ3U MINGW64 /Z
$ cd 1WT23CS000/aboutMyself
```

Now we are in the project repository. To clone this in VS Code use:

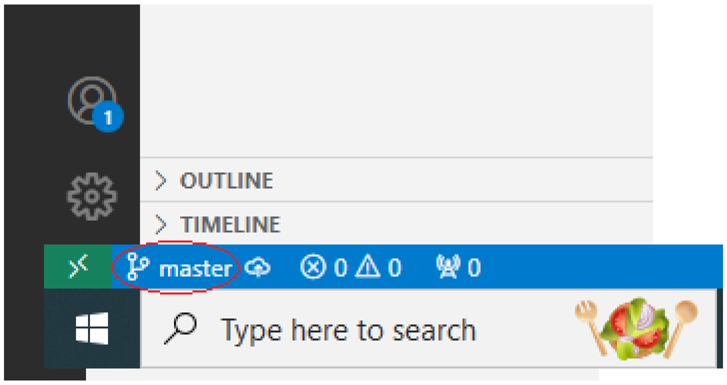
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /Z/1WT23CS000/aboutMyself (master)
$ code .
```

It will clone the repository in VS code and it will get operated as shown below:

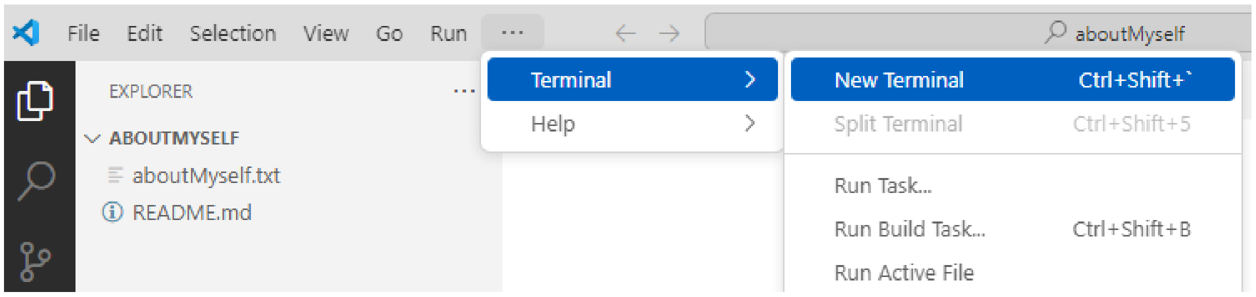


We can see the repository aboutMyself inside which there are two files available. aboutMyself.txt and README.md files.

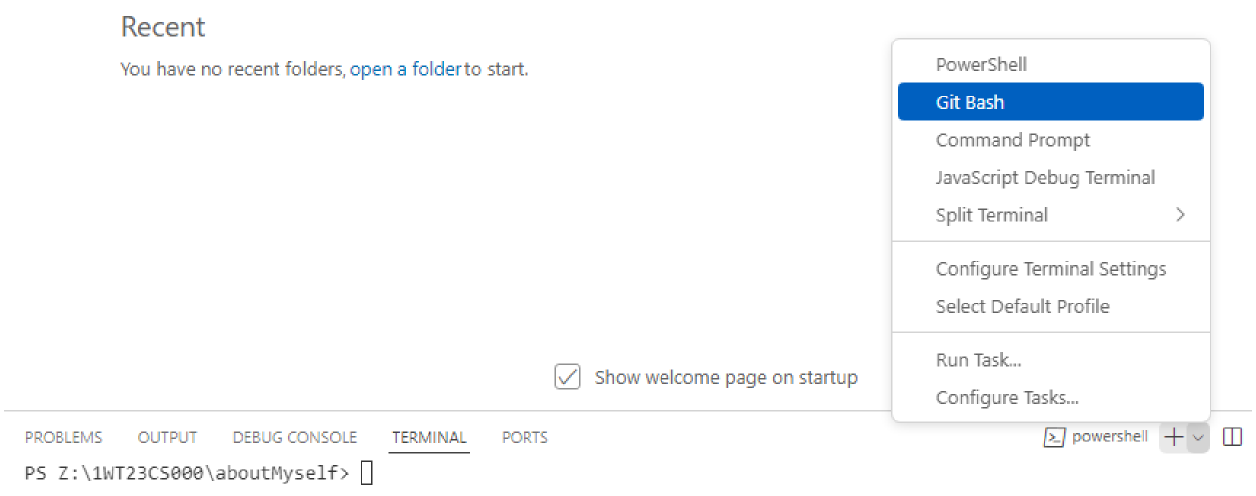
We can see on left side the repository aboutMyself. The branch will be shown on left bottom screen:



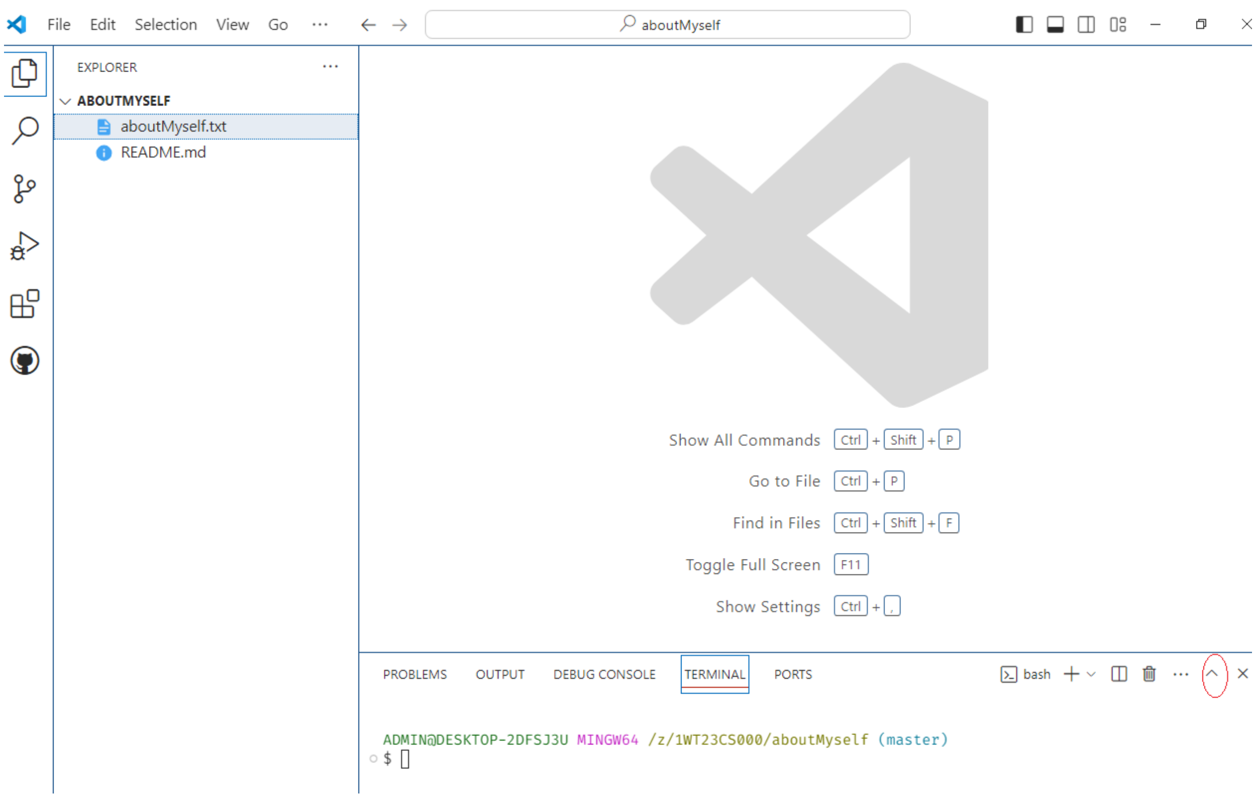
To operate git bash under VS code: Click on the Terminal then NewTerminal and then Git bash. We can see the terminal screen below.



By default it will show powershell. Change it to git Bash as shown below:

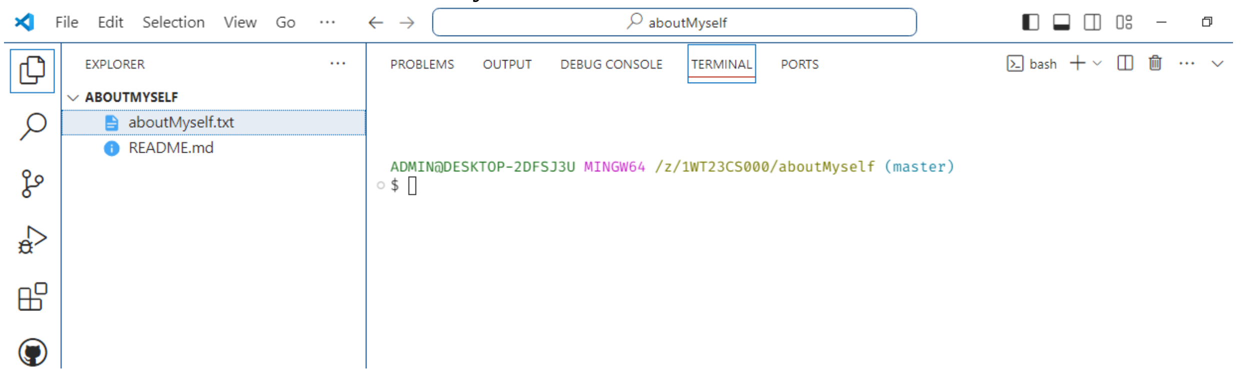

Recent
You have no recent folders, open a folder to start.
PowerShell
Git Bash
Command Prompt
JavaScript Debug Terminal
Split Terminal
Configure Terminal Settings
Select Default Profile
Run Task...
Configure Tasks...
Show welcome page on startup
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell

```
PS Z:\1WT23CS000\aboutMyself>
```


File Edit Selection View Go
aboutMyself
EXPLORER
ABOUTMYSELF
aboutMyself.txt
README.md
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Toggle Full Screen F11
Show Settings Ctrl + ,
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash

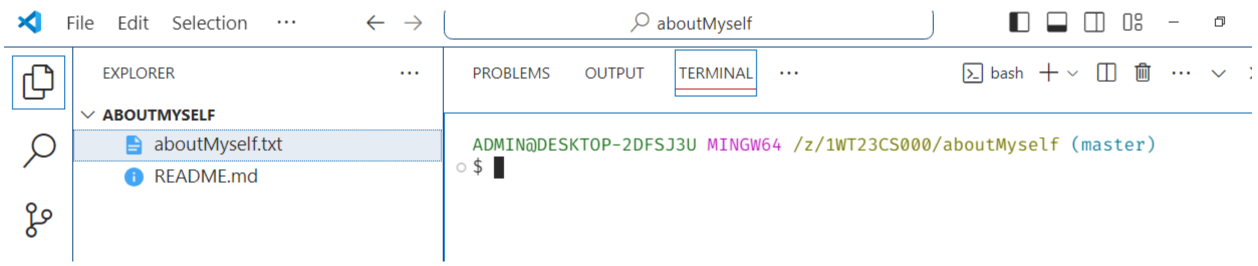
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$
```

To maximize the screen, click on ^ symbol.


File Edit Selection View Go
aboutMyself
EXPLORER
ABOUTMYSELF
aboutMyself.txt
README.md
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash

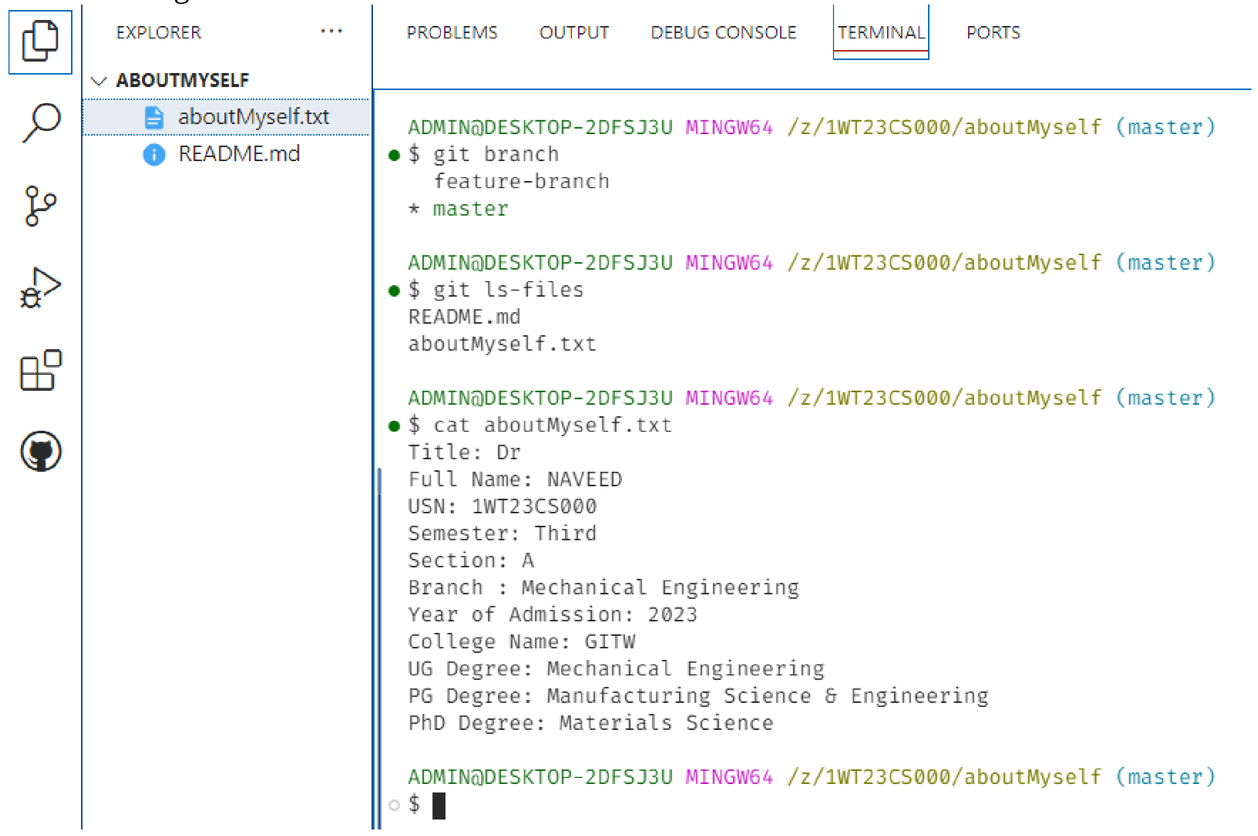
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$
```

Note: To Zoom In and Zoom OUT use Ctrl+ and Ctrl -



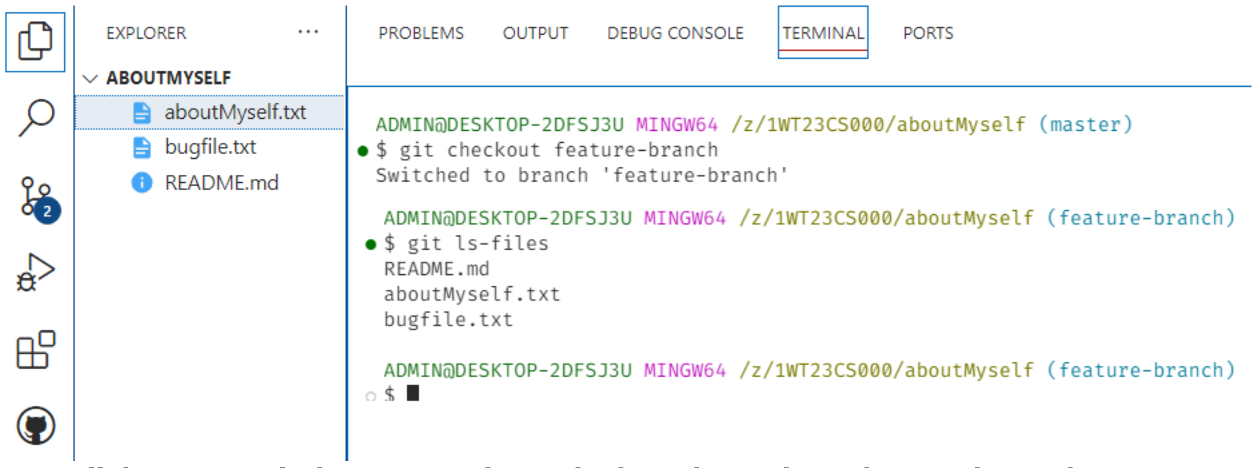
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ 
```

All the basic git commands which were used in GITBash can be used over here as shown below:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git branch
  feature-branch
* master

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git ls-files
README.md
aboutMyself.txt

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat aboutMyself.txt
Title: Dr
Full Name: NAVEED
USN: 1WT23CS000
Semester: Third
Section: A
Branch : Mechanical Engineering
Year of Admission: 2023
College Name: GITW
UG Degree: Mechanical Engineering
PG Degree: Manufacturing Science & Engineering
PhD Degree: Materials Science

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ 
```

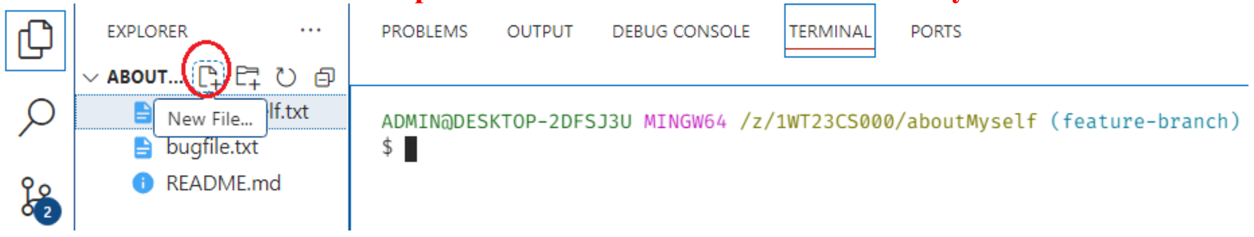
Note: If (master) is not highlighted then use git init command.



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git checkout feature-branch
Switched to branch 'feature-branch'

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git ls-files
README.md
aboutMyself.txt
bugfile.txt

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ 
```

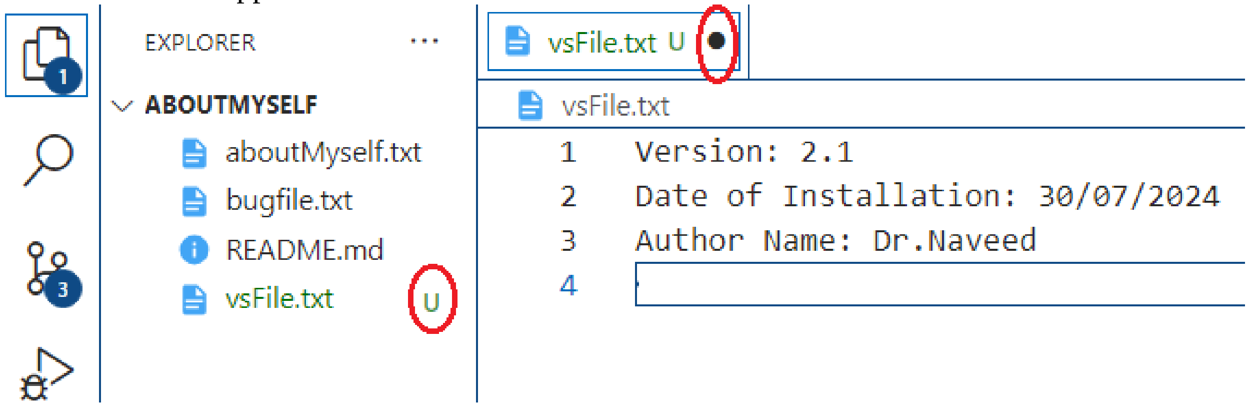
Now all the commands that were used in git bash can be used over here as shown above.

To add a new file “vsfile.txt” apart from the commands we can directly use:



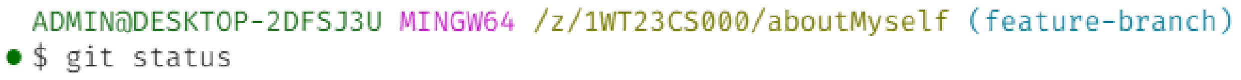
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ 
```

Add the data: Vs File details: Here we can copy and paste it. This was not available in gitbash.
Version: 2.1
Date of Installation: 30/07/2024
Author Name: Dr.Naveed
U- shows it is untracked. The white dot symbol shows it is not saved. Use Ctrl+S to save. The white dot will disappear.



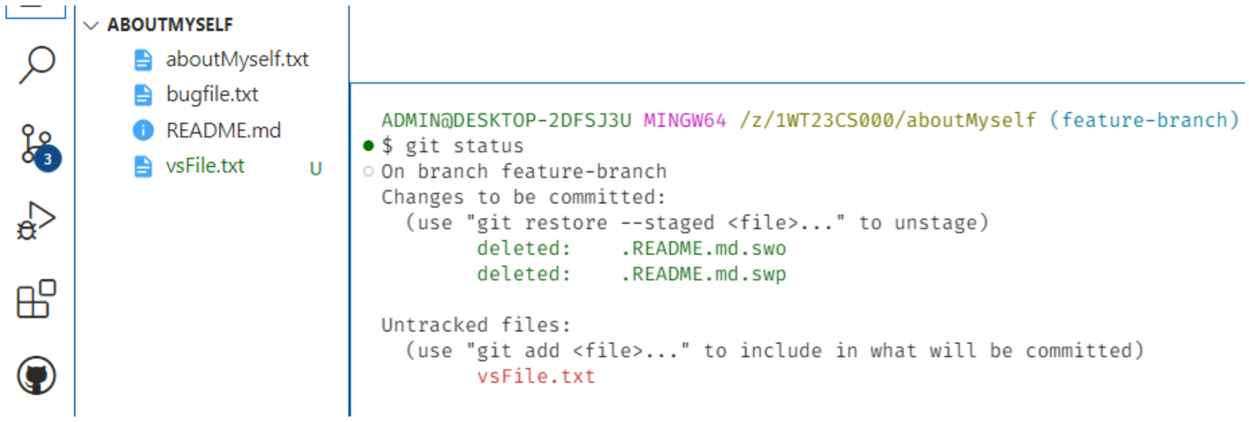
```
Version: 2.1
Date of Installation: 30/07/2024
Author Name: Dr.Naveed

```

If we check the status:



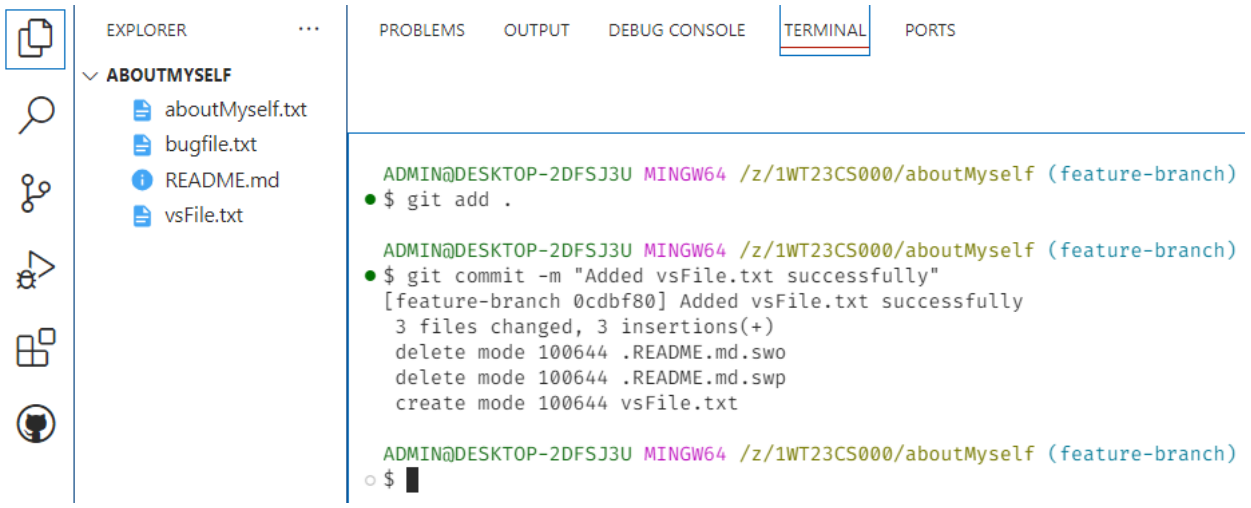
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
● $ git status
```

It shows untracked file without commit.



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
● $ git status
○ On branch feature-branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    .README.md.swo
        deleted:    .README.md.swp

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

To track the file and get added use:



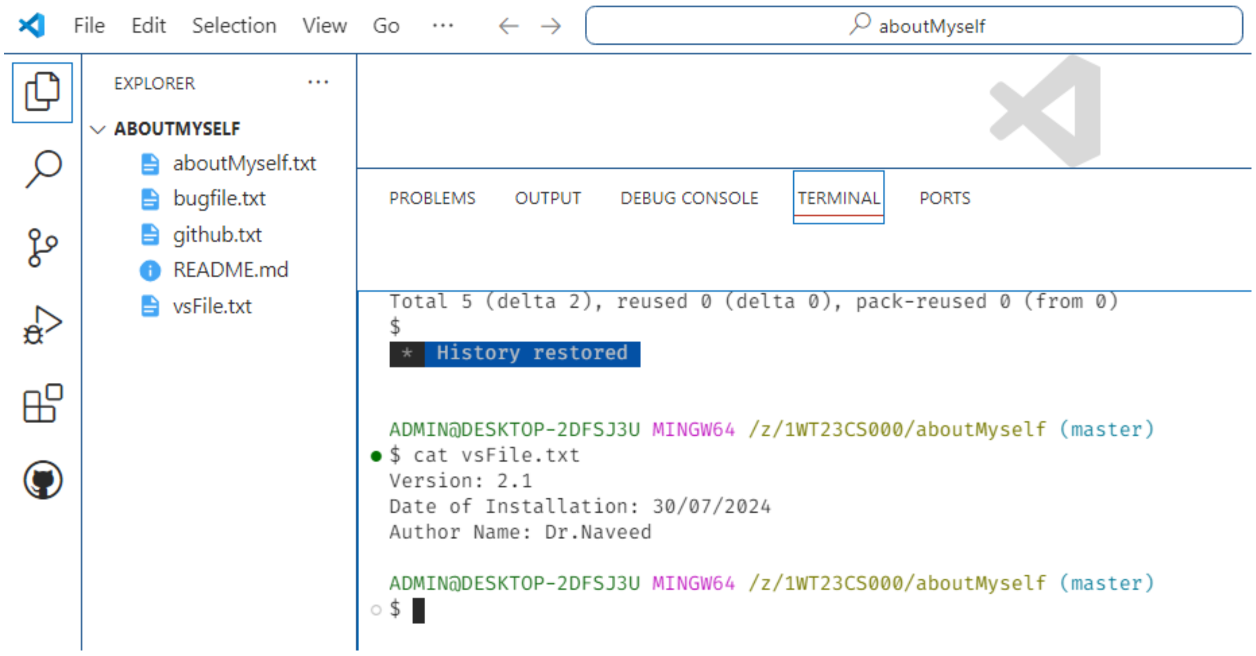
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git add .

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git commit -m "Added vsFile.txt successfully"
[feature-branch 0cdbf80] Added vsFile.txt successfully
 3 files changed, 3 insertions(+)
 delete mode 100644 .README.md.swo
 delete mode 100644 .README.md.swp
 create mode 100644 vsFile.txt

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$
```

Now there is no symbol U. If we check the status:

```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (feature-branch)
$ git status
On branch feature-branch
nothing to commit, working tree clean
```

Everything is fine.

OUTPUT:



```
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
$
 * History restored

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ cat vsFile.txt
Version: 2.1
Date of Installation: 30/07/2024
Author Name: Dr.Naveed

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$
```

Introduction to GIT-HUB

GitHub is a web-based platform for version control and collaboration, primarily used for code. It leverages Git, a distributed version control system, to manage and track changes in source code during software development. GitHub offers both free and paid plans and provides a wide range of features that facilitate collaborative coding, project management, and more.
GitHub is a powerful platform that combines the functionalities of Git version control with additional tools for collaboration, project management, and automation. Its widespread adoption in the software development community makes it an essential tool for developers, teams, and organizations.

Key Features of GitHub

1. Repository Hosting:
- GitHub allows users to host repositories, which can be either public or private. Repositories contain all the files, history, and metadata for a project.

2. Version Control with Git:
- GitHub uses Git to track changes in code. This includes features like branching, merging, pull requests, and commit history.

3. Collaboration:
- Multiple developers can work on the same project simultaneously. GitHub provides tools for managing contributions, reviewing code, and discussing changes.

4. Pull Requests:
- A pull request is a feature that allows developers to propose changes to a repository. Other team members can review, discuss, and approve or reject these changes before they are merged into the main codebase.

5. Issues and Project Management:
- GitHub provides an issue tracking system to manage bugs, feature requests, and other tasks. It also offers project boards and task management features to help organize and prioritize work.

6. Actions and Continuous Integration/Continuous Deployment (CI/CD):
- GitHub Actions allow users to automate workflows, such as running tests, building projects, and deploying applications, directly from their repositories.

7. Documentation and Wikis:
- Users can create and maintain documentation for their projects using Markdown files within the repository or GitHub's wiki feature.

8. Social Networking for Developers:
- GitHub has social features like following users, starring repositories, and forking projects, which help foster community and collaboration.

9. Security Features:
- GitHub offers various security features, such as vulnerability alerts, Dependabot for dependency management, and security advisories.

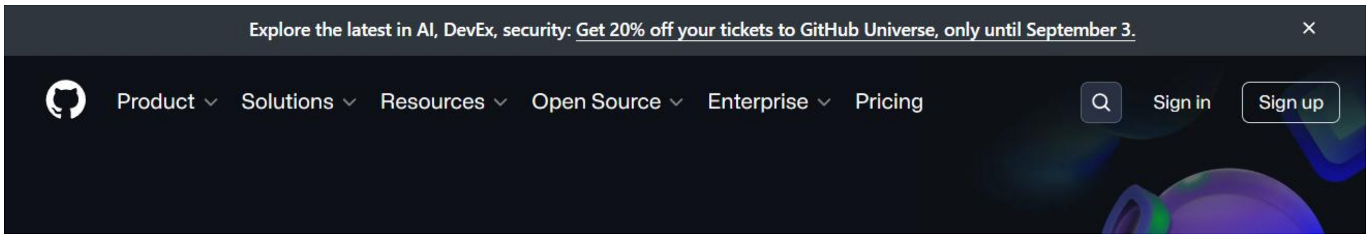
10. Integration with Other Tools:
- GitHub integrates with a wide range of third-party tools and services, including IDEs, project management tools, CI/CD systems, and more.

Applications:

1.Open-Source Projects: Many open-source projects are hosted on GitHub, allowing contributors from around the world to collaborate.
2.Private Projects: Companies and organizations use GitHub for private projects, taking advantage of its tools for collaboration, code review, and CI/CD.
3.Personal Projects and Portfolios: Developers often use GitHub to showcase their work and build a portfolio.
4.Learning and Experimentation: GitHub is a valuable resource for learning new technologies, as many projects and tutorials are hosted there.

To Sign-up and create an account in the git-hub refer the following link and follow the steps to create an account.

https://github.com/

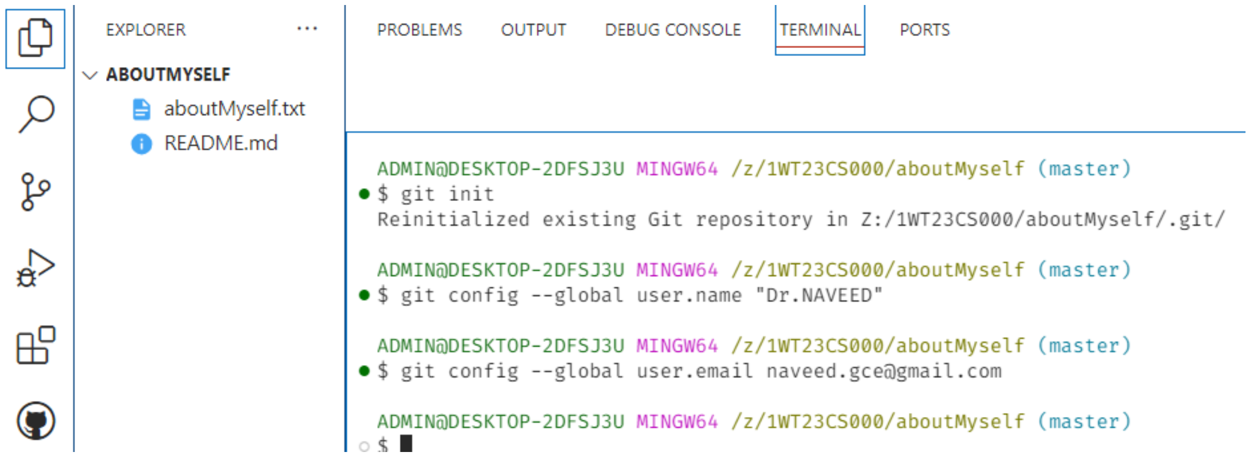

Explore the latest in AI, DevEx, security: Get 20% off your tickets to GitHub Universe, only until September 3.

Experiment-10 **VS Code and Github**

Aim:

1. To clone the repository /z/1WT23CS000/aboutMyself from VS code to your github account for both master branch and feature-branch.
2. To create a new file github.txt in VS code add the following data: Push to the github account with committed message "Added github.txt file successfully" into the master branch.
    - githubID: naveedgce
    - Version: 2.1
    - Date of installation: 01/08/2024

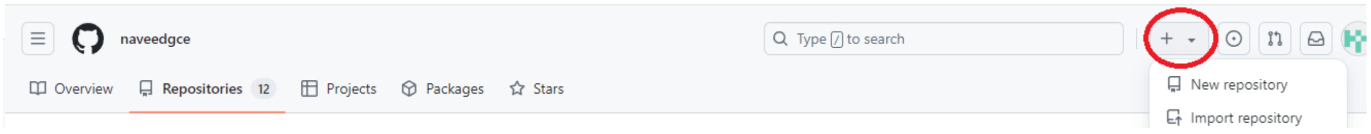
First open the VS code and initialise the repository and configure the author name and Email address as shown below:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git init
Reinitialized existing Git repository in Z:/1WT23CS000/aboutMyself/.git/

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git config --global user.name "Dr.NAVEED"

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git config --global user.email naveed.gce@gmail.com

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ 
```

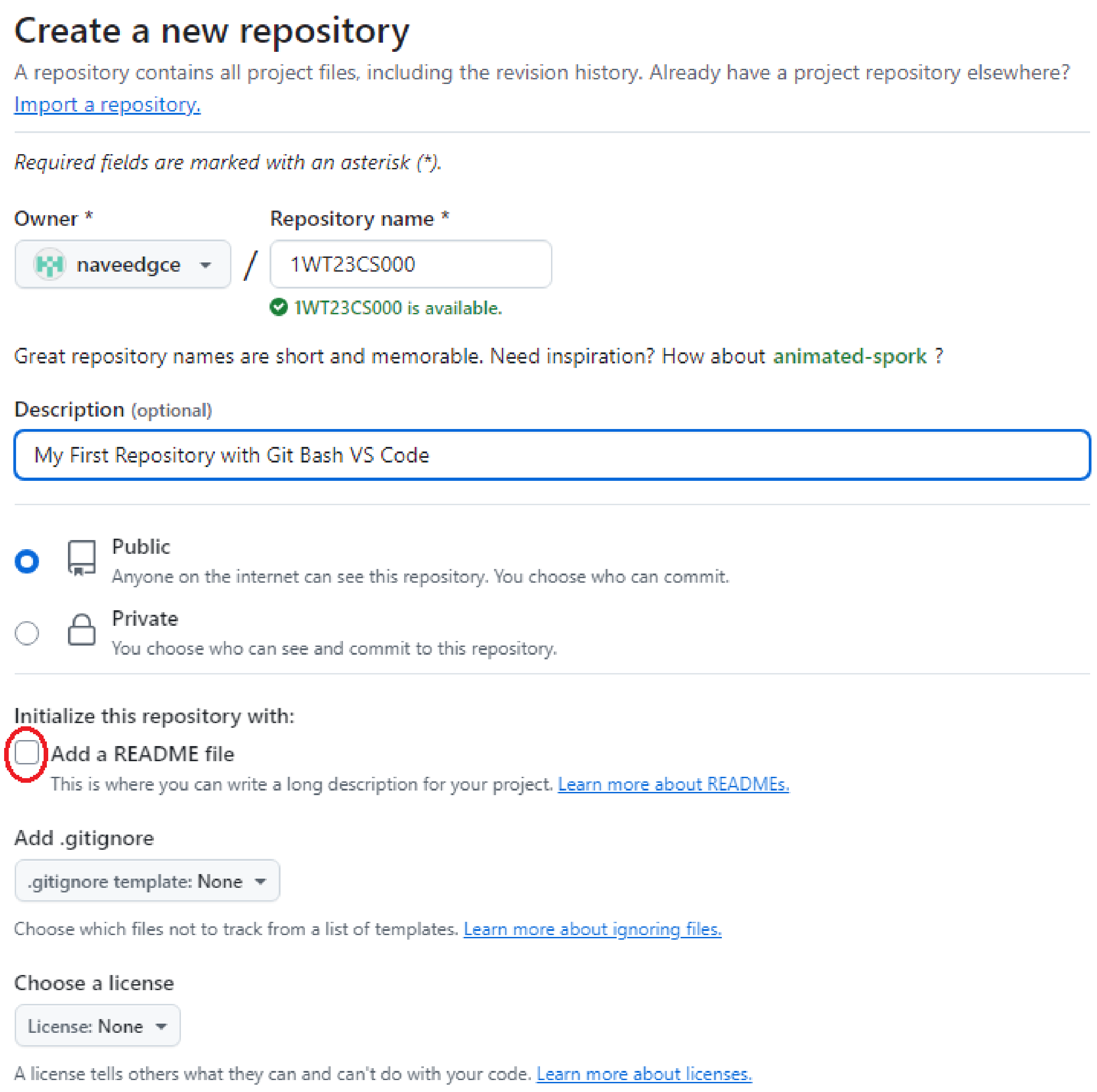
Sign in to your github account:My github account is naveedgce. You can type your own github account with your USN as ID.

Click on + sign to add New repository



Fill up all the details as shown below. Give the repository name as 1WT23CS000.Make it public so that everybody can access.

Donot activate README.md file as there is already README.md file available in our repository.


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner * naveedgce / Repository name * 1WT23CS000

1WT23CS000 is available.

Great repository names are short and memorable. Need inspiration? How about animated-spork ?

Description (optional): My First Repository with Git Bash VS Code

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

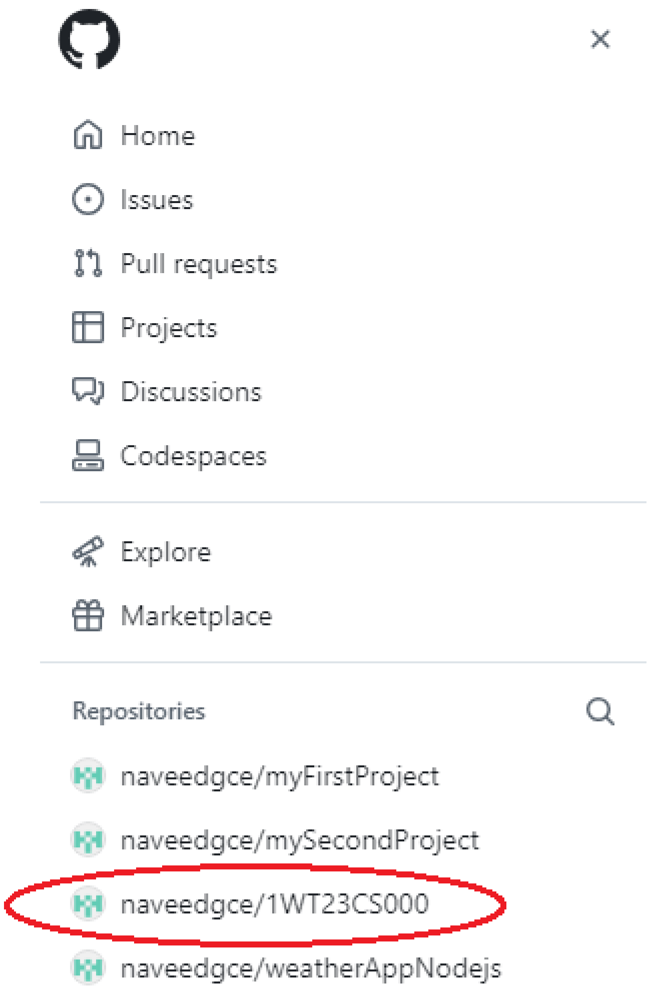
Initialize this repository with:

☐ Add a README file
This is where you can write a long description for your project. Learn more about READMEs.

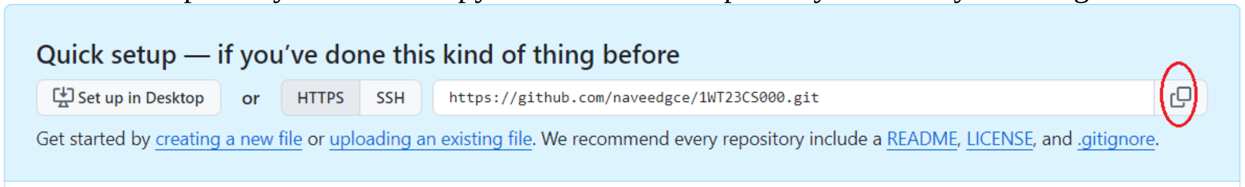
Add .gitignore: .gitignore template: None
Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license: License: None
A license tells others what they can and can't do with your code. Learn more about licenses.

Now we can see that a new repository by “1WT23CS000” is created as shown below:



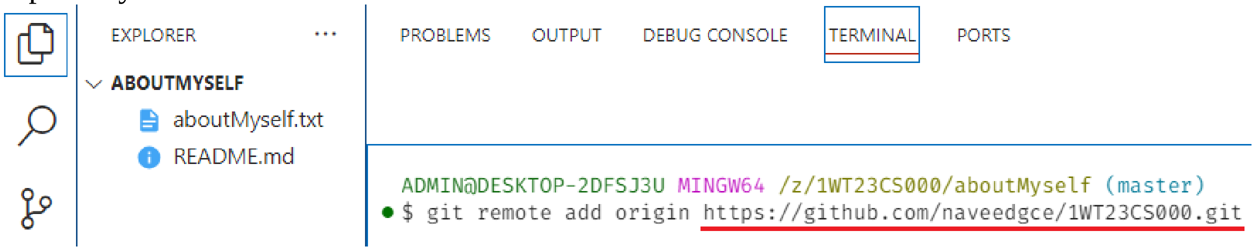
Click on the repository name and copy the URL of the repository created by referring:


Quick setup — if you've done this kind of thing before

```
https://github.com/naveedgce/1WT23CS000.git
```

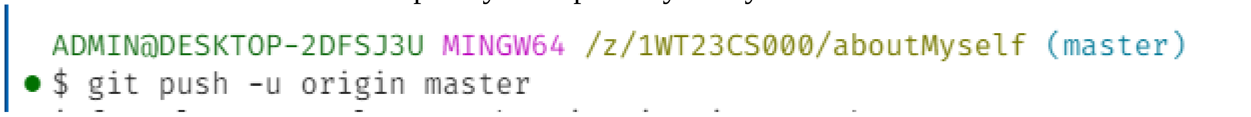
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

Use the below code in VS coding screen git remote add origin < paste the copied URL of github repository >:



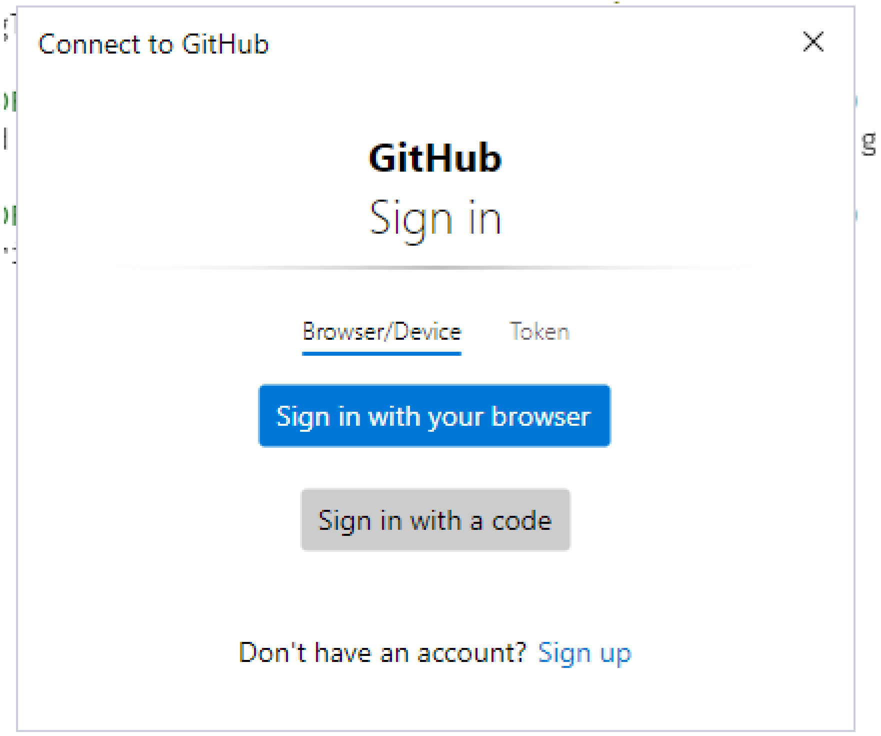
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git remote add origin https://github.com/naveedgce/1WT23CS000.git
```

After this use the below code to push your repository into your GitHub account:

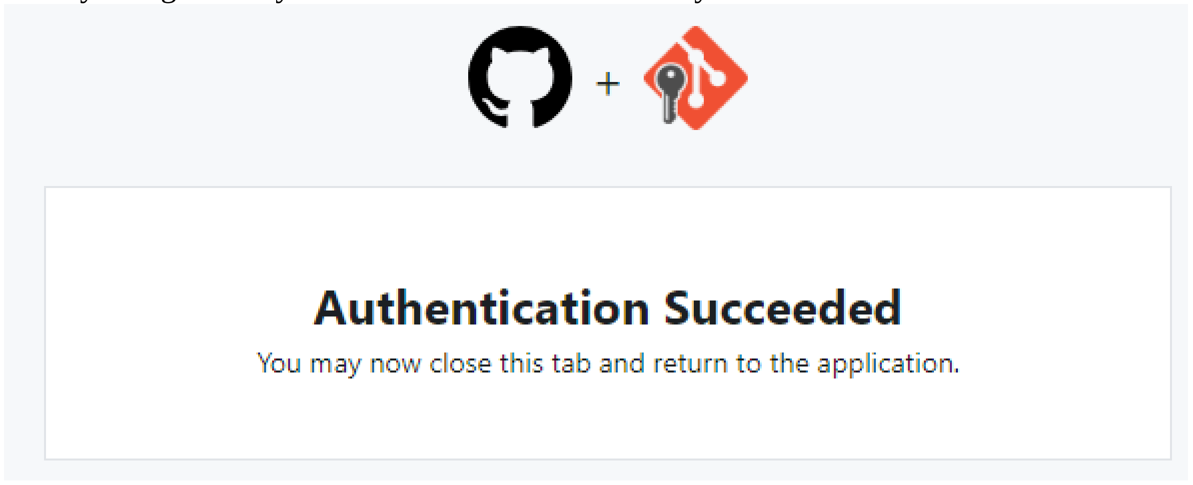


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git push -u origin master
```

It will ask for Authentication by signing into your GitHub account:



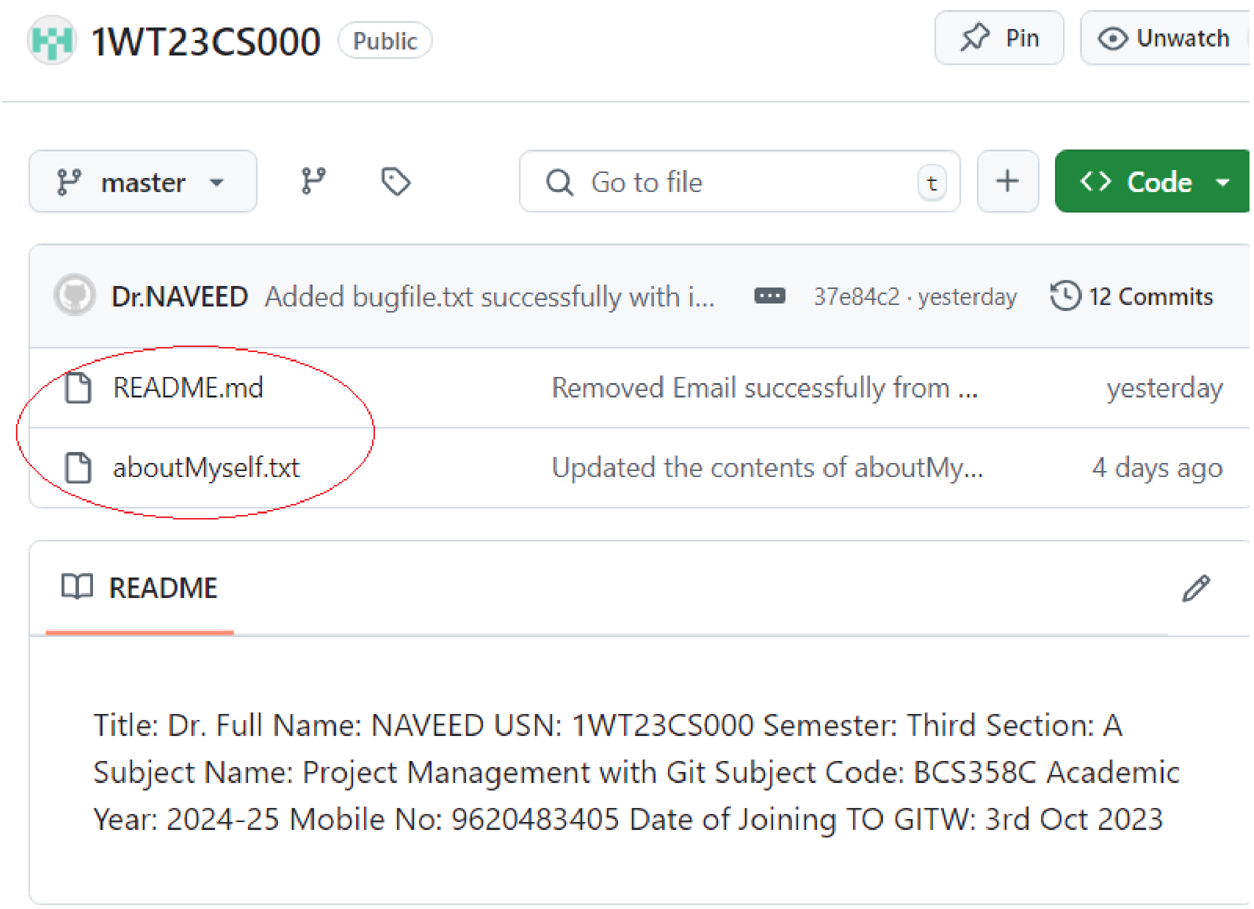
Once you log in into your GitHub account successfully it will show:



Authentication Succeeded

You may now close this tab and return to the application.

Now we can find all the files copied into our GitHub repository;


1WT23CS000 Public

Pin Unwatch

master

Go to file t

Code

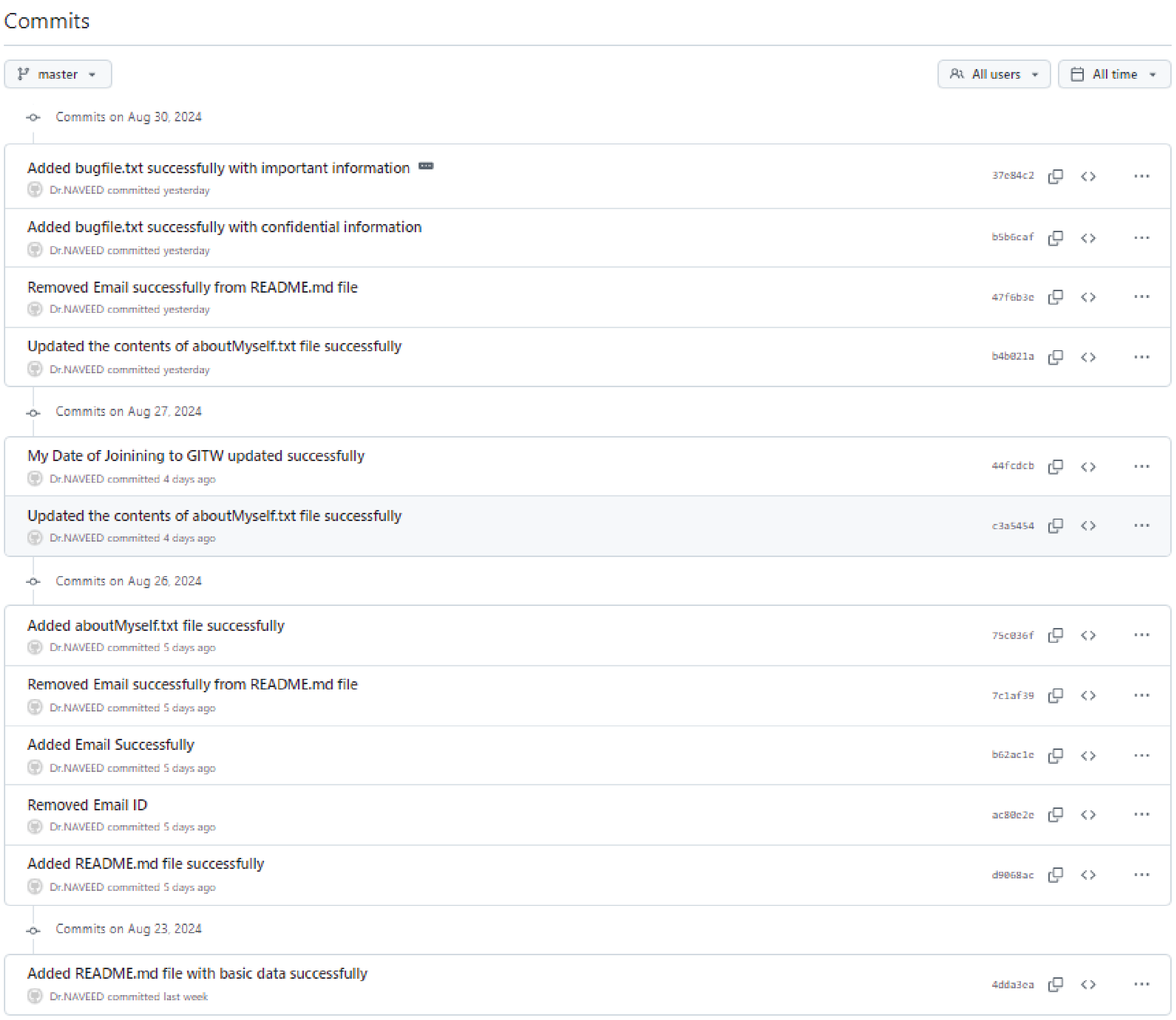
Dr.NAVEED Added bugfile.txt successfully with i... 37e84c2 · yesterday 12 Commits

README.md	Removed Email successfully from ...	yesterday
aboutMyself.txt	Updated the contents of aboutMy...	4 days ago

README

Title: Dr. Full Name: NAVEED USN: 1WT23CS000 Semester: Third Section: A Subject Name: Project Management with Git Subject Code: BCS358C Academic Year: 2024-25 Mobile No: 9620483405 Date of Joining TO GITW: 3rd Oct 2023

It shows 12 commits done so far. If we click on it:


Commits

master · All users · All time

Commits on Aug 30, 2024

Added bugfile.txt successfully with important information
Dr.NAVEED committed yesterday · 37e84c2

Added bugfile.txt successfully with confidential information
Dr.NAVEED committed yesterday · b5b6caf

Removed Email successfully from README.md file
Dr.NAVEED committed yesterday · 47f6b3e

Updated the contents of aboutMyself.txt file successfully
Dr.NAVEED committed yesterday · b4b021a

Commits on Aug 27, 2024

My Date of Joinining to GITW updated successfully
Dr.NAVEED committed 4 days ago · 44fcdcb

Updated the contents of aboutMyself.txt file successfully
Dr.NAVEED committed 4 days ago · c3a5454

Commits on Aug 26, 2024

Added aboutMyself.txt file successfully
Dr.NAVEED committed 5 days ago · 75c036f

Removed Email successfully from README.md file
Dr.NAVEED committed 5 days ago · 7c1af39

Added Email Successfully
Dr.NAVEED committed 5 days ago · b62ac1e

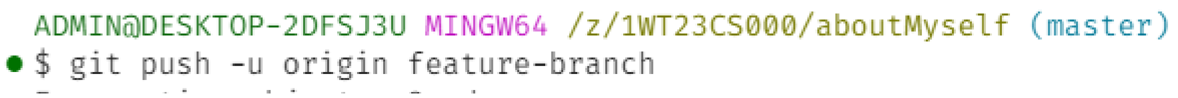
Removed Email ID
Dr.NAVEED committed 5 days ago · ac80e2e

Added README.md file successfully
Dr.NAVEED committed 5 days ago · d9068ac

Commits on Aug 23, 2024

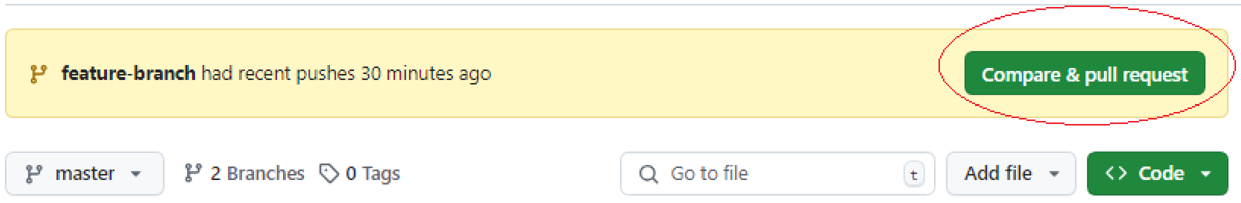
Added README.md file with basic data successfully
Dr.NAVEED committed last week · 4dda3ea

To push feature-branch:



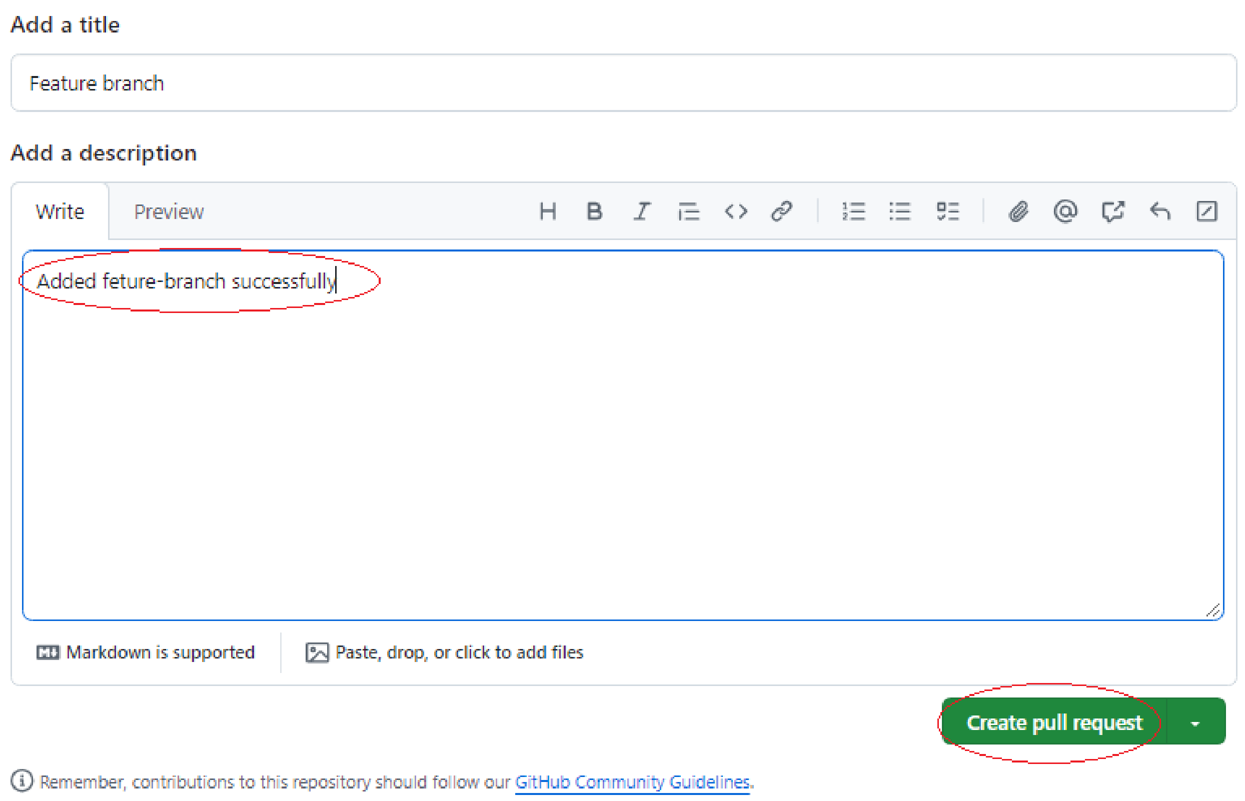
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git push -u origin feature-branch
```

It will add into the GitHub account and it will ask to compare and pull the request as shown below:



feature-branch had recent pushes 30 minutes ago

Click on compare & Pull request:


Add a title

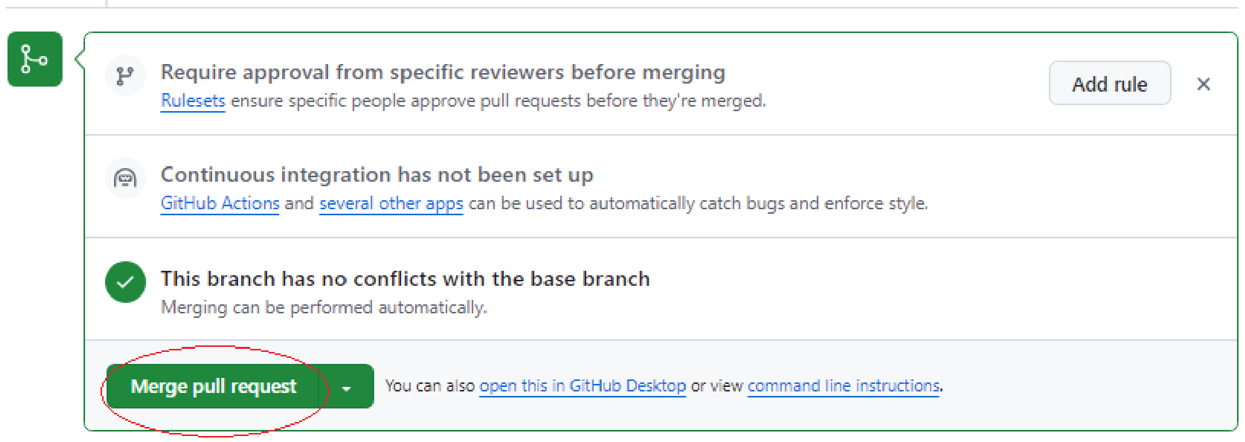
Feature branch

Add a description

Added feture-branch successfully

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Click on Create pull request:


Require approval from specific reviewers before merging

Rulesets ensure specific people approve pull requests before they're merged.

Continuous integration has not been set up

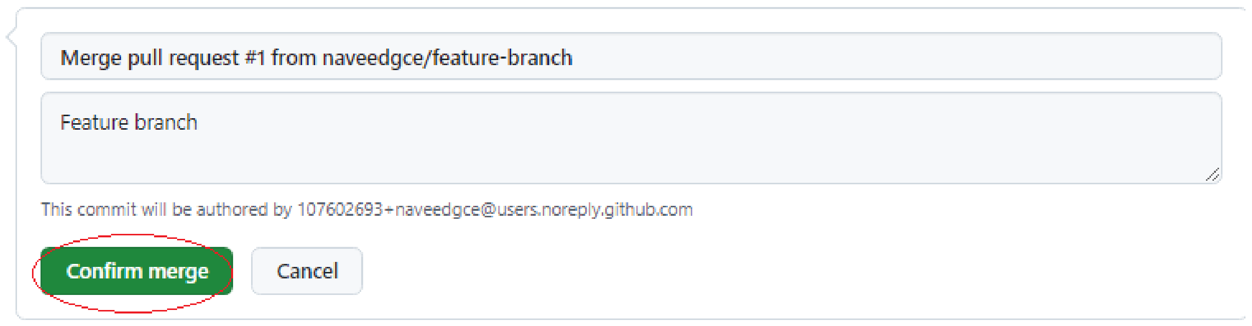
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch

Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

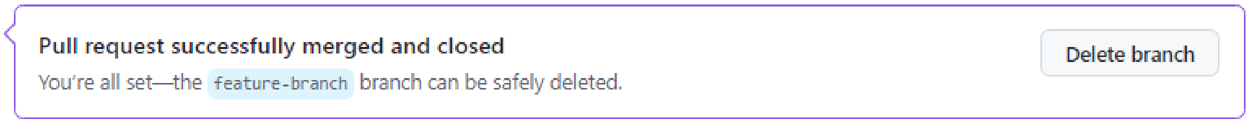
Click on Merge Pull Request.


Merge pull request #1 from naveedgce/feature-branch

Feature branch

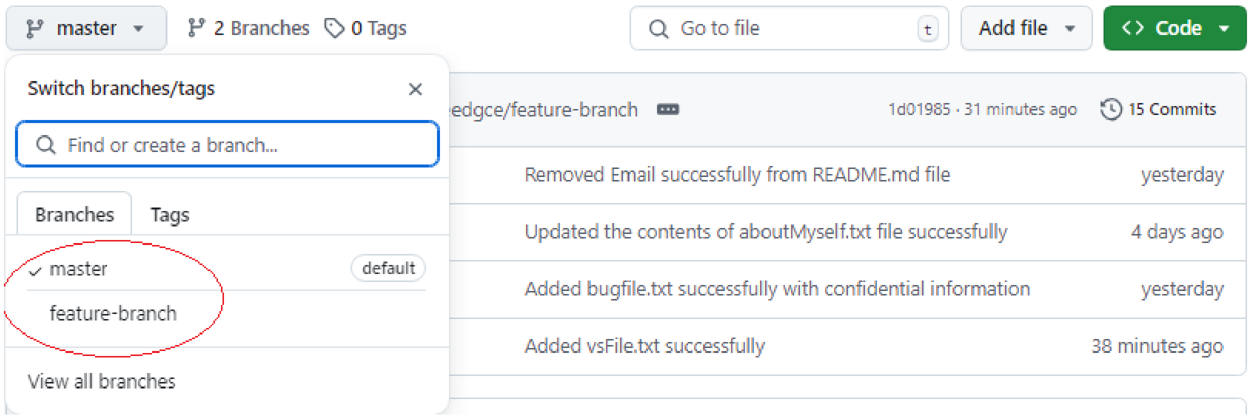
This commit will be authored by 107602693+naveedgce@users.noreply.github.com

It will ask for Confirm merge. Click on it.


Pull request successfully merged and closed

You're all set—the `feature-branch` branch can be safely deleted.

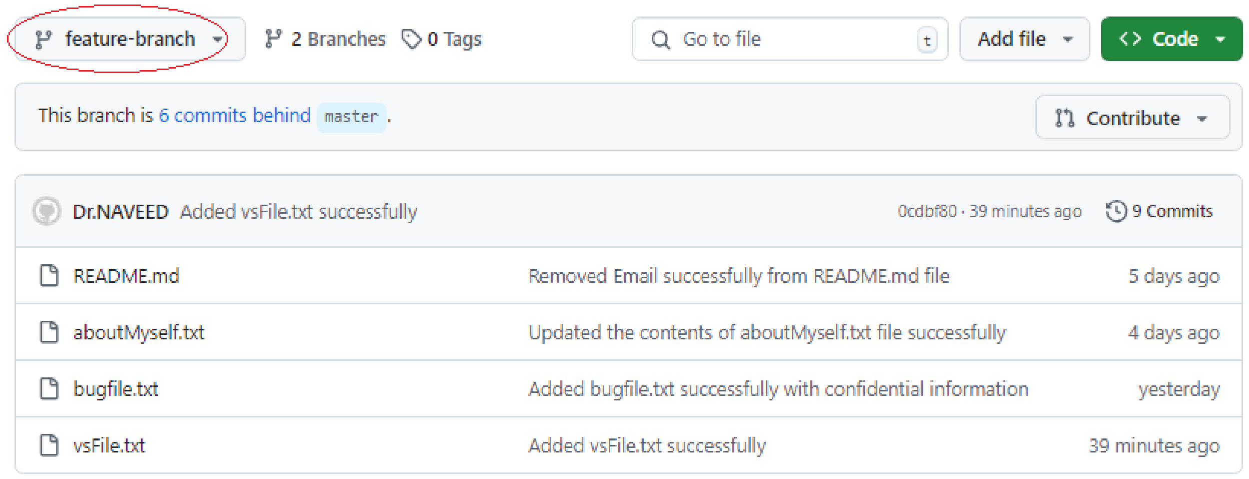
It shows a message of successfully merged and closed. Now if we click on the branches, it will show two branches.



edgce/feature-branch 1d01985 · 31 minutes ago 15 Commits

Removed Email successfully from README.md file	yesterday
Updated the contents of aboutMyself.txt file successfully	4 days ago
Added bugfile.txt successfully with confidential information	yesterday
Added vsFile.txt successfully	38 minutes ago

If we click on the feature-branch:



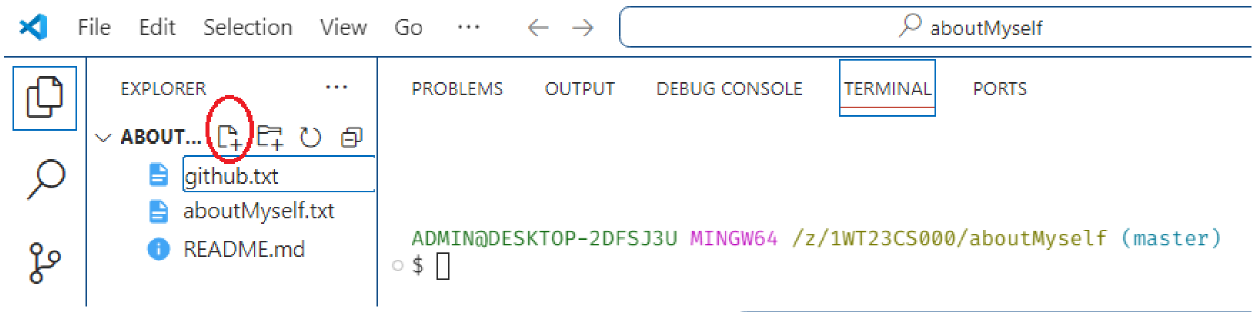
This branch is 6 commits behind `master`.

Dr.NAVEED Added vsFile.txt successfully 0cdbf80 · 39 minutes ago 9 Commits

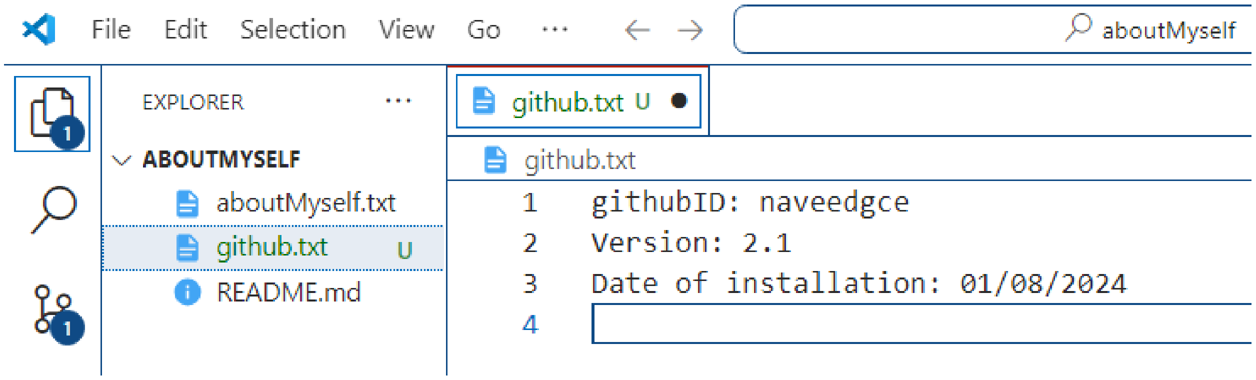
README.md	Removed Email successfully from README.md file	5 days ago
aboutMyself.txt	Updated the contents of aboutMyself.txt file successfully	4 days ago
bugfile.txt	Added bugfile.txt successfully with confidential information	yesterday
vsFile.txt	Added vsFile.txt successfully	39 minutes ago

It shows all the files that are available in feature-branch along with the 9 commits.

To add a new file github.txt into the VS code and then to push into the GitHub account Use:
Click on New file and enter the data into it and save it.

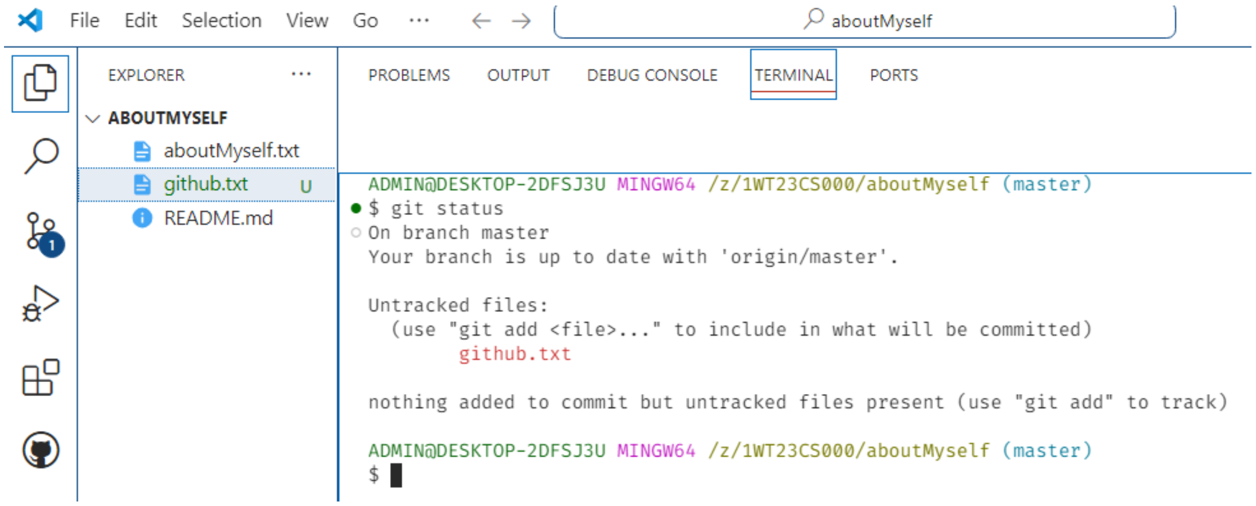


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ 
```



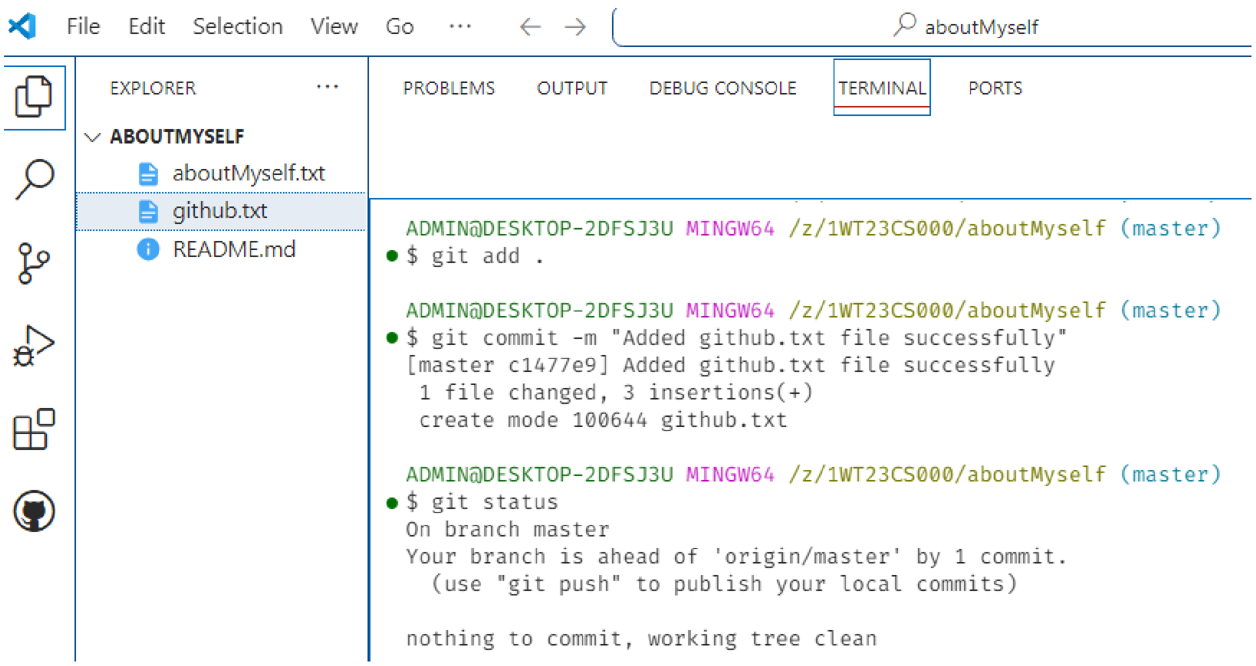
```
githubID: naveedgce
Version: 2.1
Date of installation: 01/08/2024
```

If we check the status:



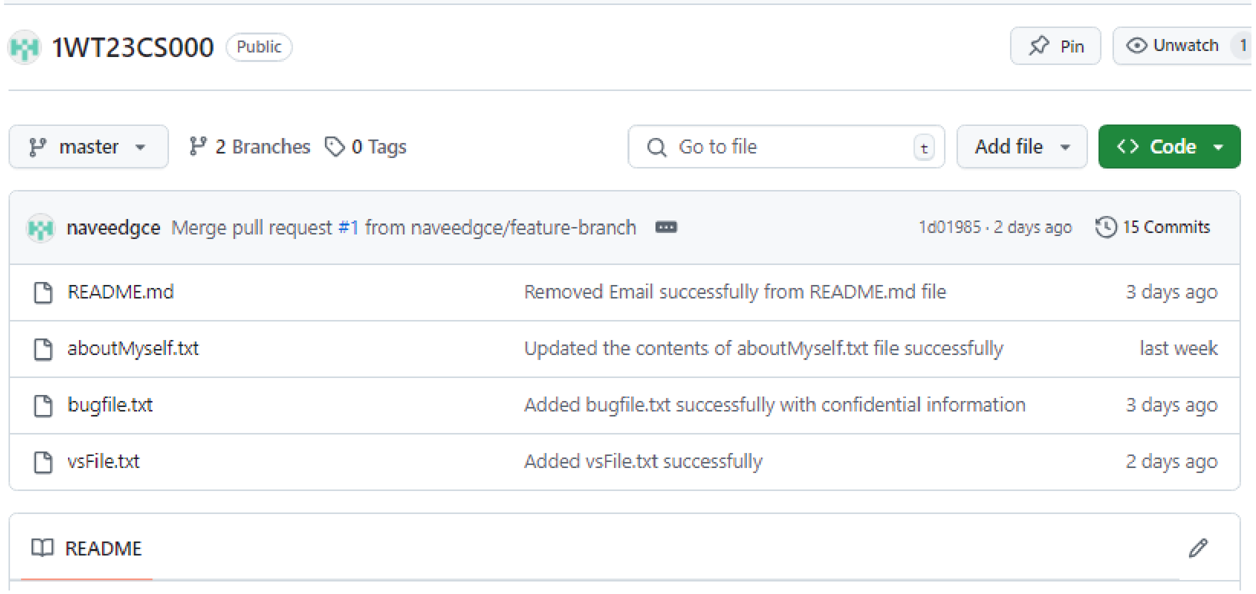
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

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

nothing added to commit but untracked files present (use "git add" to track)

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ 
```

It shows untracked file with symbol U. To track it and to commit use:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git add .

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git commit -m "Added github.txt file successfully"
[master c1477e9] Added github.txt file successfully
 1 file changed, 3 insertions(+)
 create mode 100644 github.txt

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Check the GitHub account:

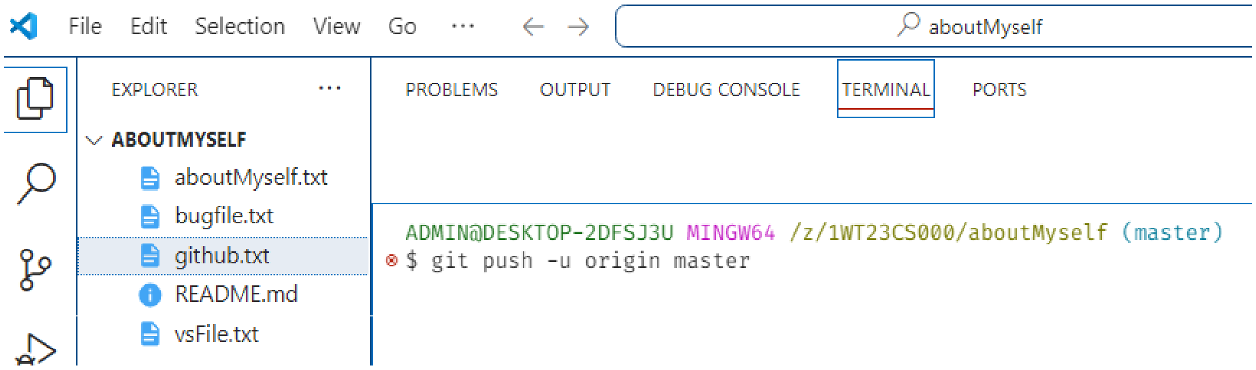


naveedgce Merge pull request #1 from naveedgce/feature-branch 1d01985 · 2 days ago 15 Commits

README.md	Removed Email successfully from README.md file	3 days ago
aboutMyself.txt	Updated the contents of aboutMyself.txt file successfully	last week
bugfile.txt	Added bugfile.txt successfully with confidential information	3 days ago
vsFile.txt	Added vsFile.txt successfully	2 days ago

There is no file added into GitHub account.

To push into GitHub account use:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git push -u origin master
```

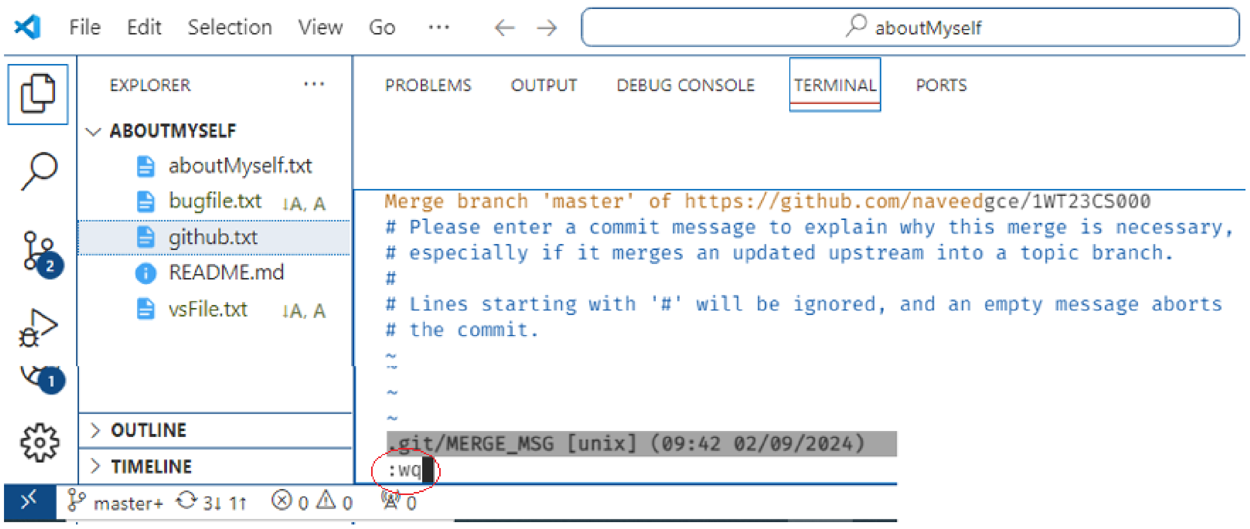
It shows some error:

```
To https://github.com/naveedgce/1WT23CS000.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/naveedgce/1WT23CS000.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

There might be some conflict. To resolve use:

```
● ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
  $ git pull origin master
```

A screen will open showing the conflict message. Keep it as such or it can be modified as well. To save and quit type ":wq" as shown below and enter.

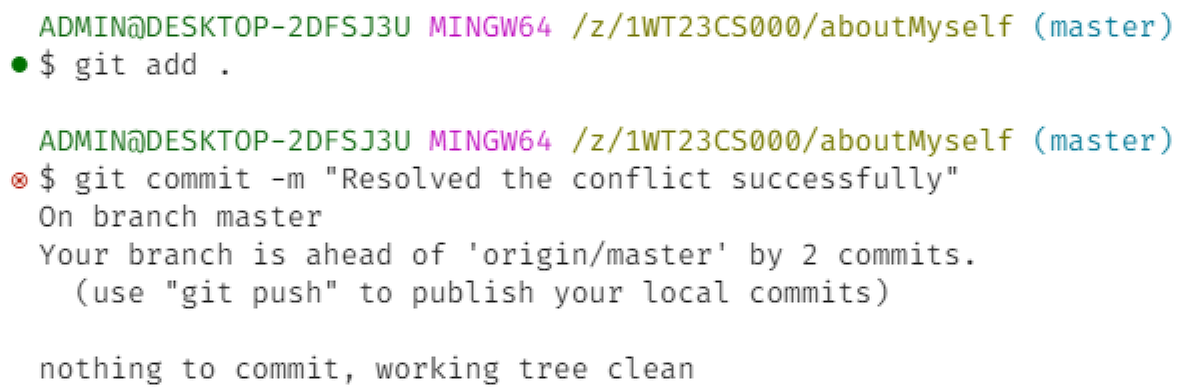


```
Merge branch 'master' of https://github.com/naveedgce/1WT23CS000
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
~
.git/MERGE_MSG [unix] (09:42 02/09/2024)
:wq
```

It shows:

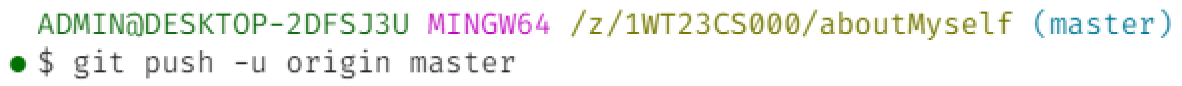
```
● ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
  $ git pull origin master
  From https://github.com/naveedgce/1WT23CS000
   * branch            master     -> FETCH_HEAD
  Merge made by the 'ort' strategy.
   bugfile.txt | 1 +
   vsFile.txt  | 3 +++
   2 files changed, 4 insertions(+)
   create mode 100644 bugfile.txt
   create mode 100644 vsFile.txt
```

Then use the following to commit with a message.



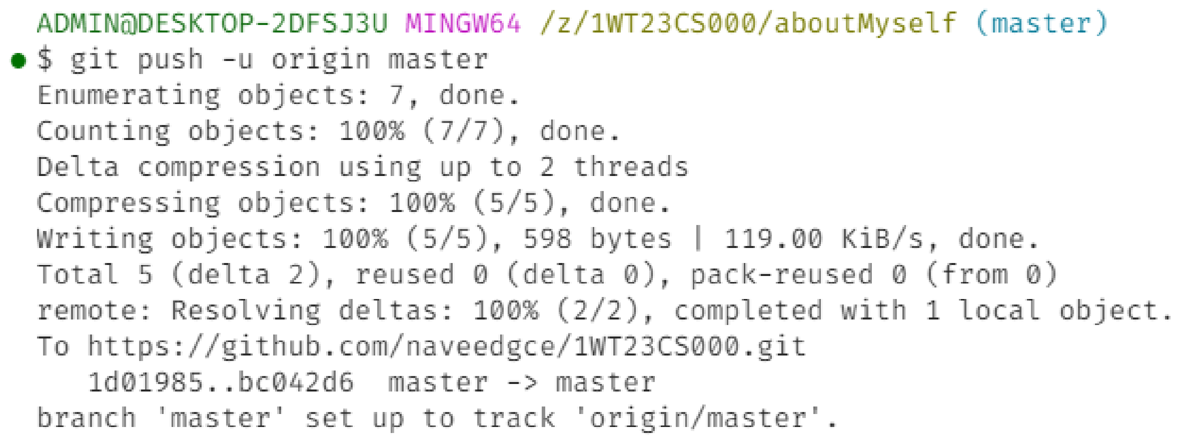
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git add .

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git commit -m "Resolved the conflict successfully"
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Now let us use the previous code to push the file into the GitHub account.



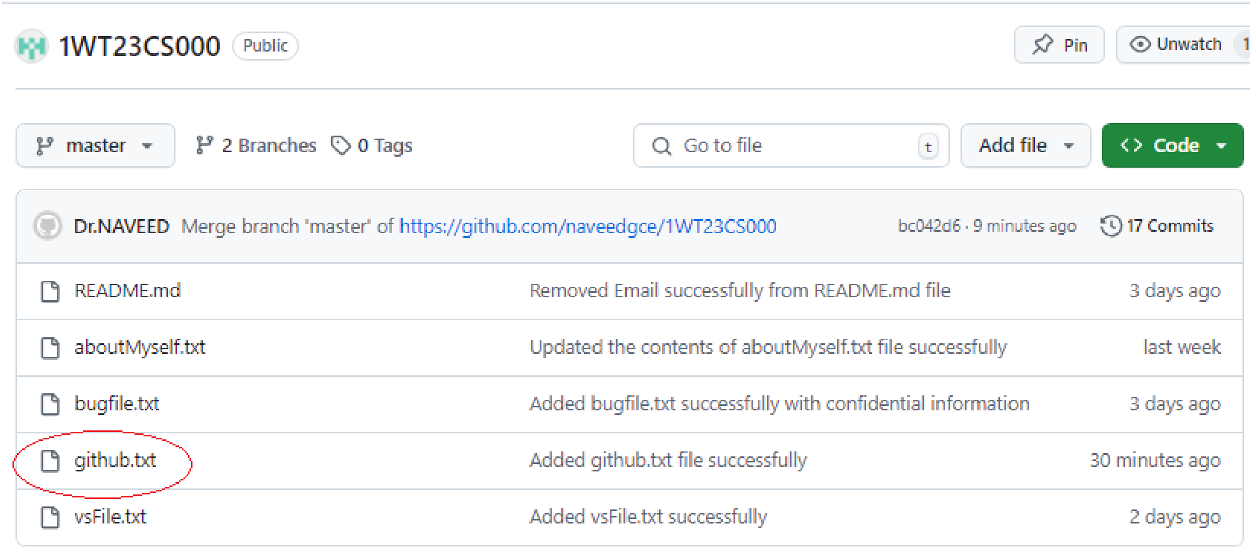
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git push -u origin master
```

Now it shows a successful message.



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/1WT23CS000/aboutMyself (master)
$ git push -u origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 598 bytes | 119.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/naveedgce/1WT23CS000.git
   1d01985..bc042d6  master -> master
branch 'master' set up to track 'origin/master'.
```

Now if we check our GitHub account:

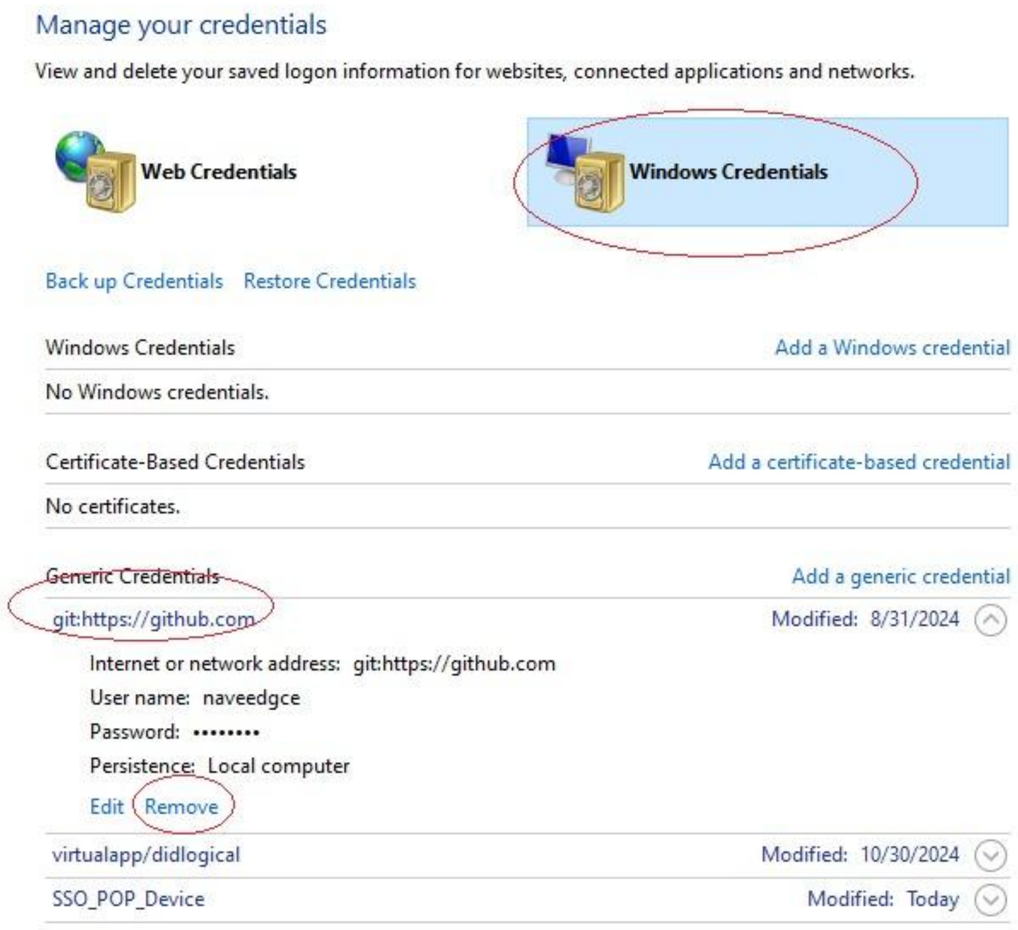


Dr.NAVEED Merge branch 'master' of https://github.com/naveedgce/1WT23CS000 bc042d6 · 9 minutes ago 17 Commits

README.md	Removed Email successfully from README.md file	3 days ago
aboutMyself.txt	Updated the contents of aboutMyself.txt file successfully	last week
bugfile.txt	Added bugfile.txt successfully with confidential information	3 days ago
github.txt	Added github.txt file successfully	30 minutes ago
vsFile.txt	Added vsFile.txt successfully	2 days ago

We can see the github.txt file added into it.

Note: If there is some conflict due to multi users . To remove the user logged into the account refer the following.


Manage your credentials

View and delete your saved logon information for websites, connected applications and networks.

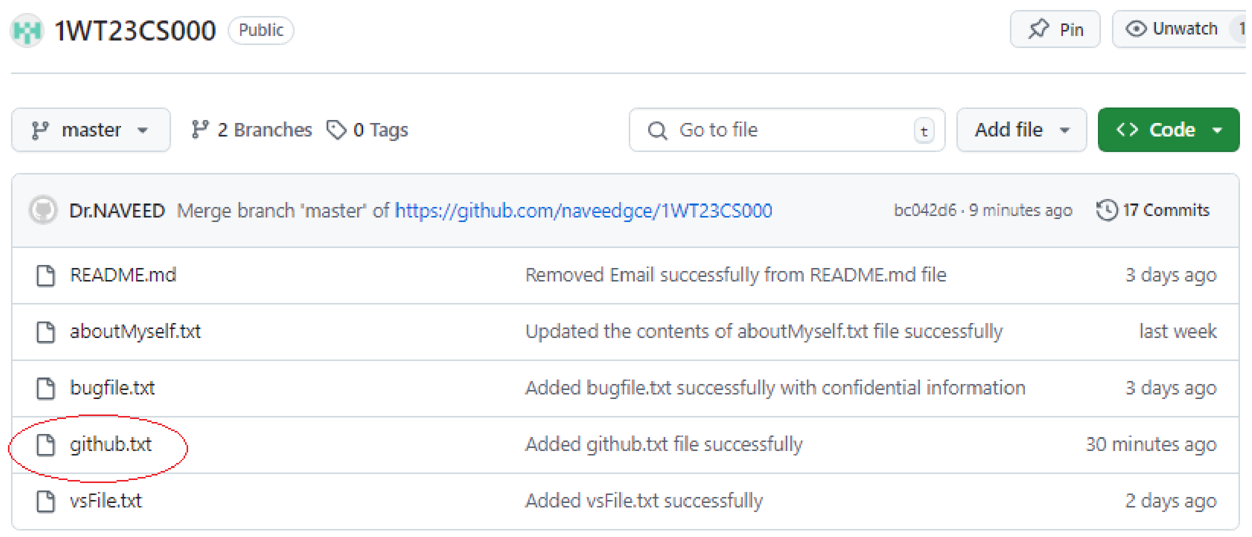
Windows Credentials	Add a Windows credential
No Windows credentials.	

Certificate-Based Credentials	Add a certificate-based credential
No certificates.	

Generic Credentials	Add a generic credential
git:https://github.com	Modified: 8/31/2024
virtualapp/didlogical	Modified: 10/30/2024
SSO_POP_Device	Modified: Today

Internet or network address: git:https://github.com
User name: naveedgce
Password: ••••••••
Persistence: Local computer

OUTPUT:


1WT23CS000 Public

Dr.NAVEED Merge branch 'master' of https://github.com/naveedgce/1WT23CS000 bc042d6 · 9 minutes ago 17 Commits

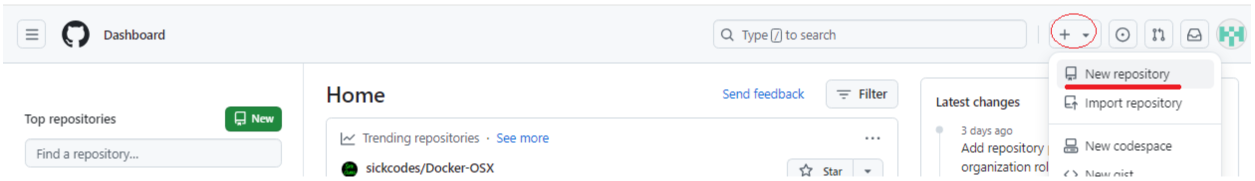
README.md	Removed Email successfully from README.md file	3 days ago
aboutMyself.txt	Updated the contents of aboutMyself.txt file successfully	last week
bugfile.txt	Added bugfile.txt successfully with confidential information	3 days ago
github.txt	Added github.txt file successfully	30 minutes ago
vsFile.txt	Added vsFile.txt successfully	2 days ago

Experiment-11 VS code and Github

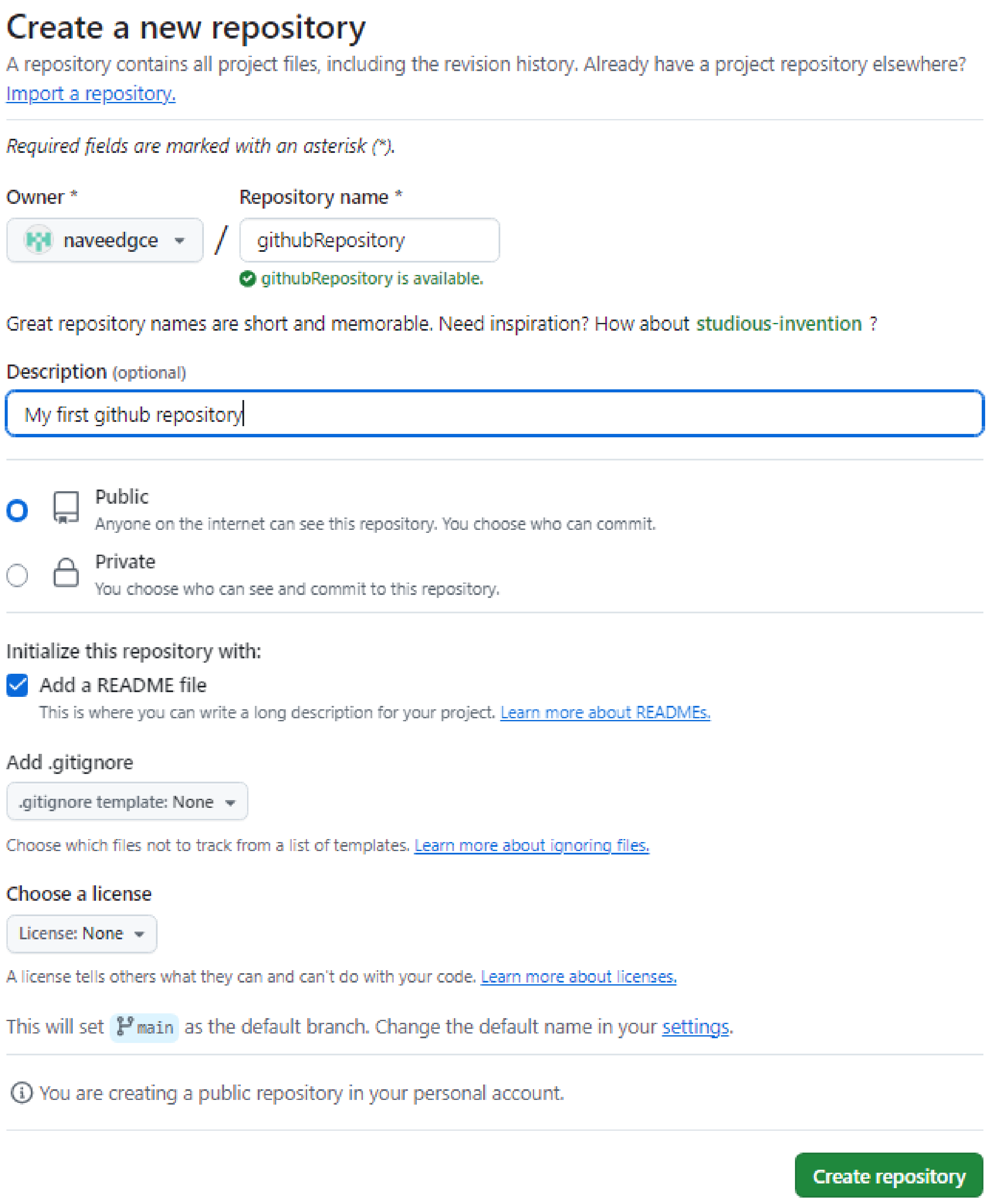
Aim:

1. To create a new repository “githubRepository” from the GitHubaccount. Add README.md file
2. To clone the repository into the VS code. To add the following data into README.md file
   - First Name:
   - Last Name:
   - Email ID
   - GitHub ID:
   - Mobile Number:
3. To push the repository into the GitHub account

First login into your GitHub account. Go to the home page and then click on add a new repository as shown below:



Type the repository name and check for the availability. Each repository is unique and cannot be of the same name. You can type your last three digits of your USN along with this name. Provide a brief description about it. My first GitHub repository. Add a README.md file which can be later filled.


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner * naveedgce / Repository name * githubRepository

githubRepository is available.

Great repository names are short and memorable. Need inspiration? How about studious-invention ?

Description (optional)

My first github repository

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with:

[✓] Add a README file
This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license

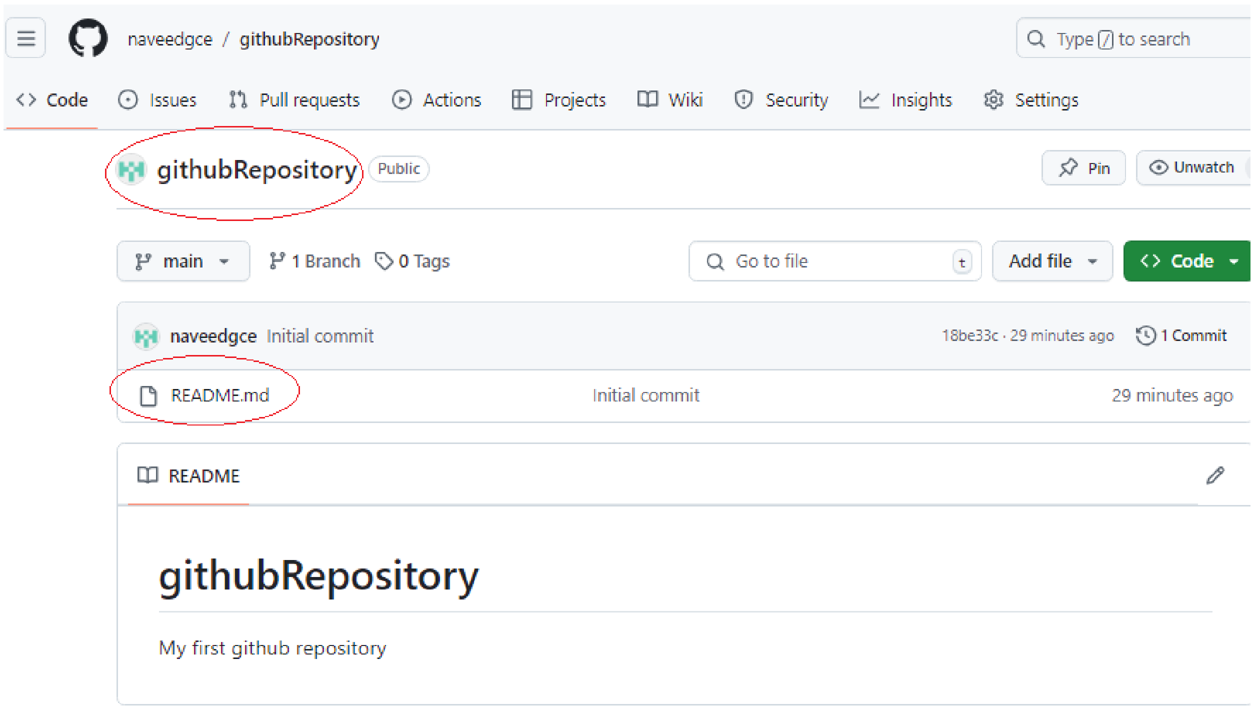
License: None

A license tells others what they can and can't do with your code. Learn more about licenses.

This will set `main` as the default branch. Change the default name in your settings.

You are creating a public repository in your personal account.

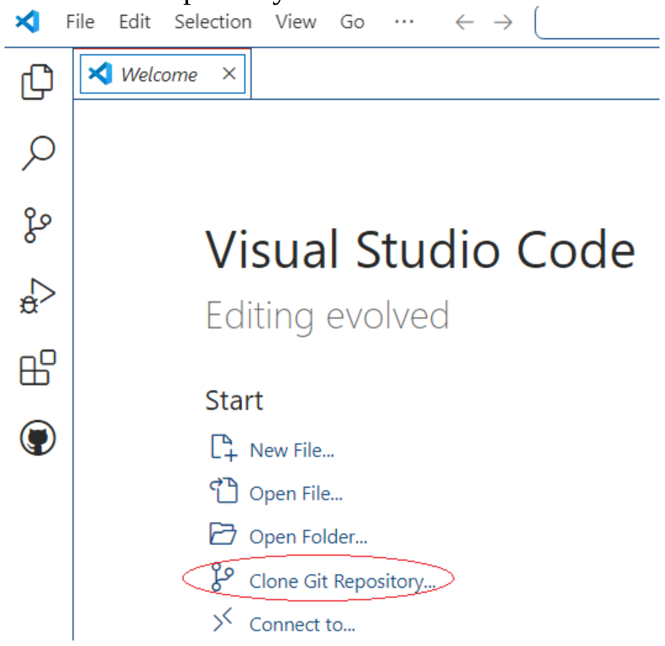
Click on Create repository.
We can see that the new repository is created in our GitHub account as shown below.



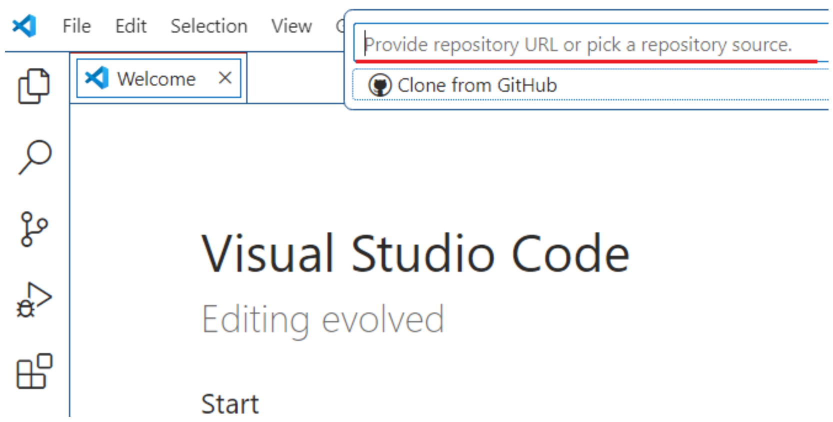
githubRepository

My first github repository

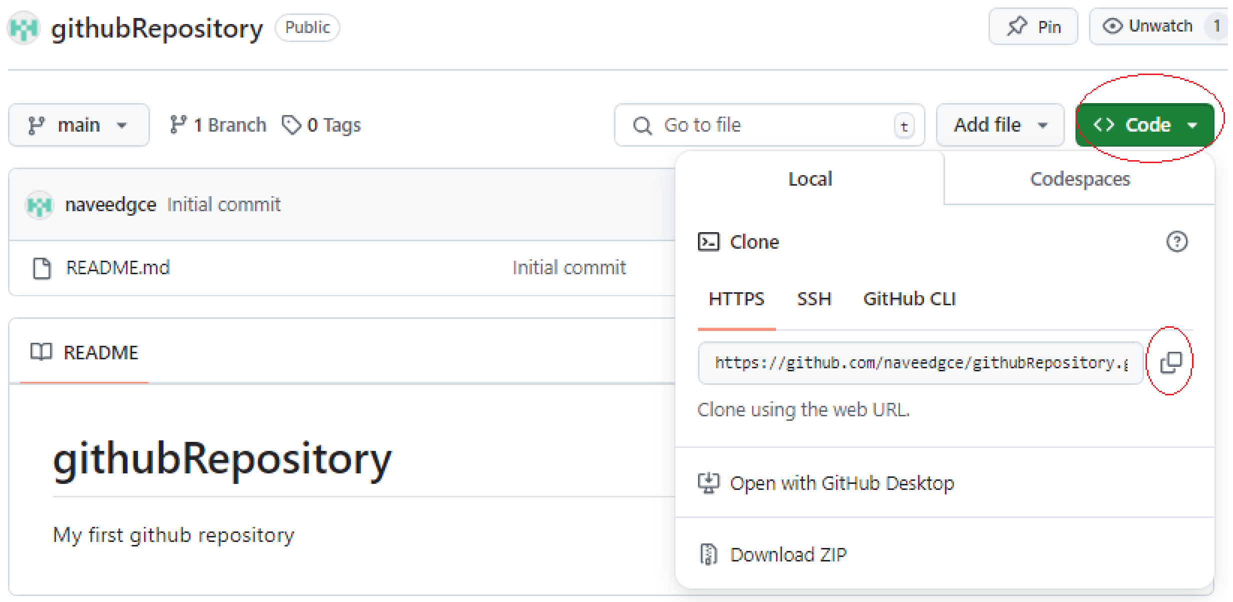
To clone this repository into the local server through VS coding platform there are two options:
Option-01:
Close the vs code editor if previous repository is in active state. Go the home page with Welcome file. Click on Clone Git Repository.



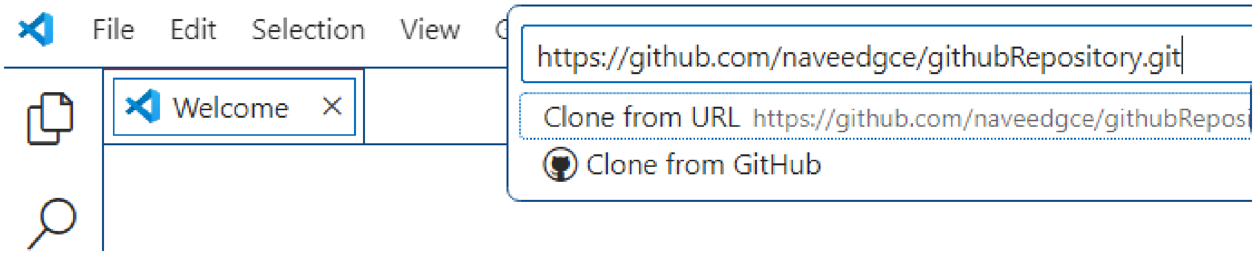
It will ask for the GitHubrepository URL.



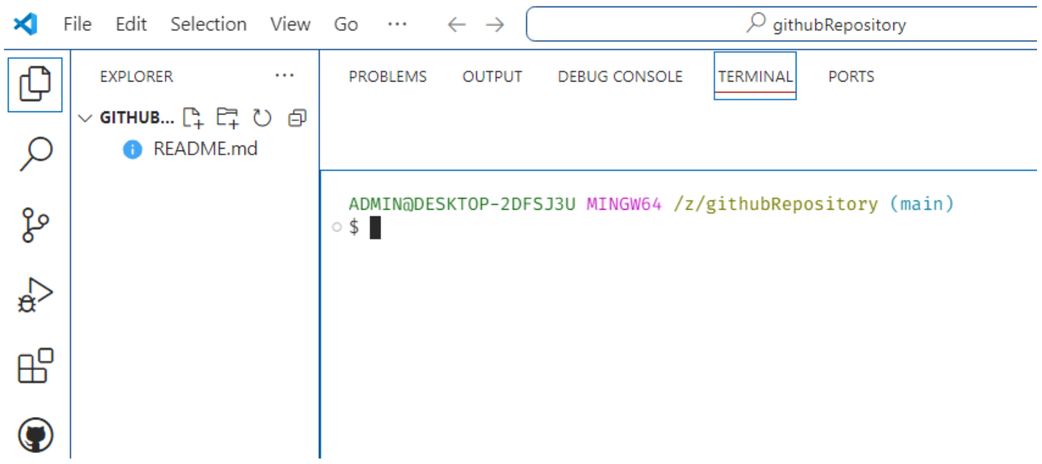
Click on code and then click on copy option of the URL as shown below:



Now go to the VS code page and add the above URL


File
Edit
Selection
View
https://github.com/naveedgce/githubRepository.git
Clone from URL https://github.com/naveedgce/githubReposi
Clone from GitHub
Welcome

It will ask for the folder. Select any drive such as Z drive.
Now we can see our repository is created in VS code platform

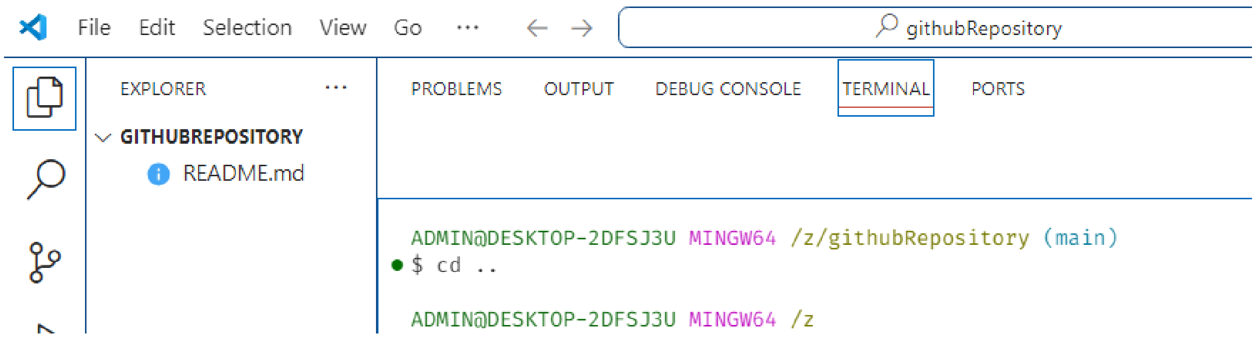


```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/githubRepository (main)
$ 
```

Now the project is ready to be worked.

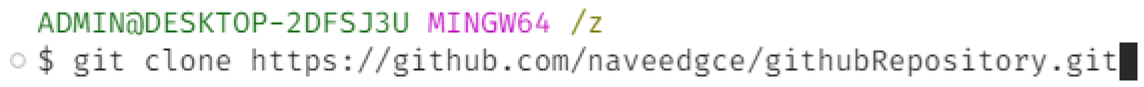
Option-02

We can also do in the other way as well. In the previous project screen: Get back to Z drive by using:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/githubRepository (main)
$ cd ..

ADMIN@DESKTOP-2DFSJ3U MINGW64 /z
```

It will get shifted to Z drive. Now we can use a code such as "git clone <paste the URL of GitHub repository>" as shown below:



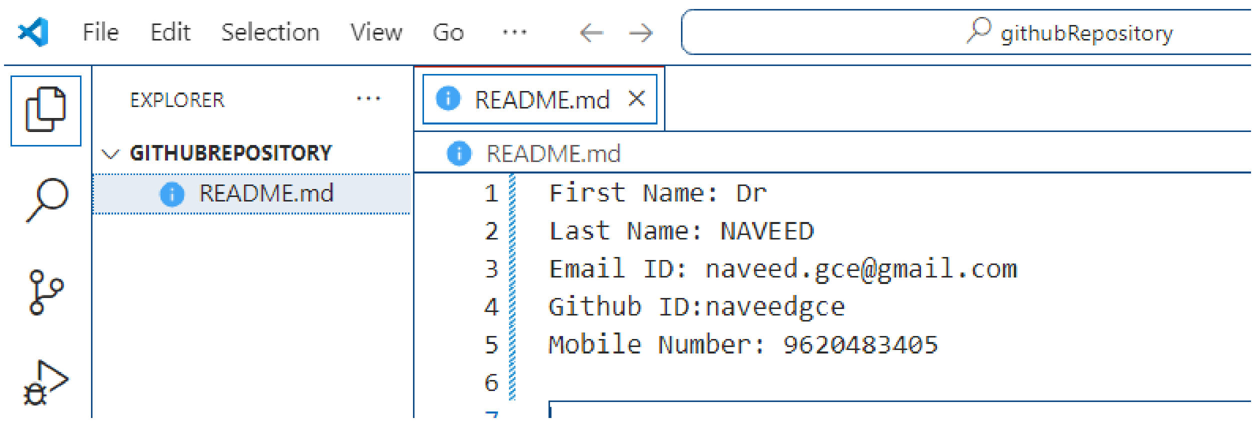
```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z
$ git clone https://github.com/naveedgce/githubRepository.git
```

The above https://........was copied from the GitHub account.

To add the data:

Open the README.md file and add the following data and save it:

1. First Name:
2. Last Name:
3. Email ID
4. GitHub ID:
5. Mobile Number:



```
First Name: Dr
Last Name: NAVEED
Email ID: naveed.gce@gmail.com
Github ID:naveedgce
Mobile Number: 9620483405
```

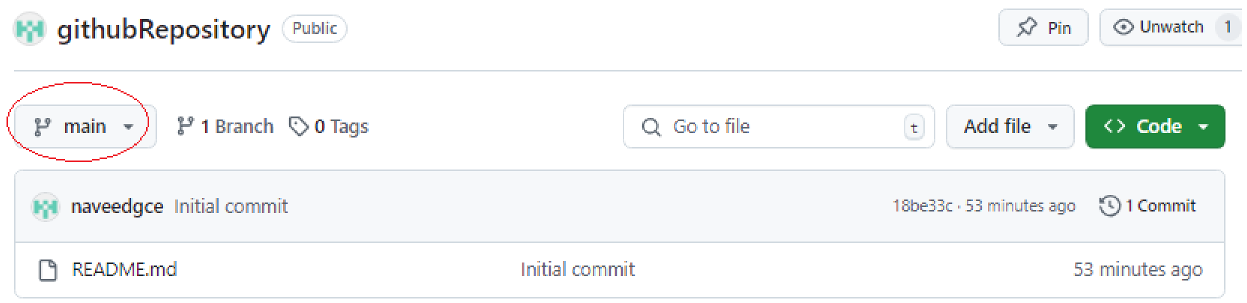
To push the updated file into the GitHub account:
First, we have to commit the task with a message. If we check the status.

```
  ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/githubRepository (main)
● $ 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

  no changes added to commit (use "git add" and/or "git commit -a")
```

Therefore, to commit use the following:

```
  ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/githubRepository (main)
● $ git add .

  ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/githubRepository (main)
● $ git commit -m "Updated README.md file successfully"
  [main 2fde9fd] Updated README.md file successfully
   1 file changed, 6 insertions(+), 2 deletions(-)

  ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/githubRepository (main)
● $ git status
  On branch main
  Your branch is ahead of 'origin/main' by 1 commit.
    (use "git push" to publish your local commits)

  nothing to commit, working tree clean
```

To push the above into the GitHub repository: Remember the repository that was created in the GitHub account shows main branch not master.



naveedgce Initial commit 18be33c · 53 minutes ago 1 Commit

README.md Initial commit 53 minutes ago

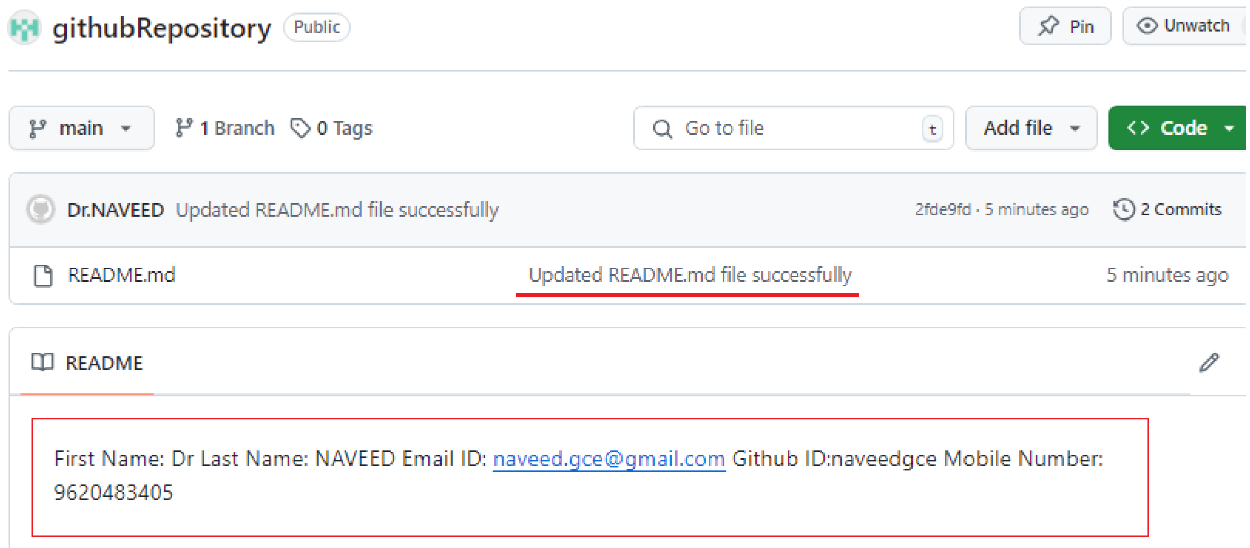
We cannot use master here for coding;

```
  ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/githubRepository (main)
● $ git push -u origin main
```

If no conflict exists then it will show a successful message.

```
  ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/githubRepository (main)
● $ git push -u origin main
  Enumerating objects: 5, done.
  Counting objects: 100% (5/5), done.
  Delta compression using up to 2 threads
  Compressing objects: 100% (2/2), done.
  Writing objects: 100% (3/3), 361 bytes | 361.00 KiB/s, done.
  Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
  To https://github.com/naveedgce/githubRepository.git
     18be33c..2fde9fd  main -> main
  branch 'main' set up to track 'origin/main'.
```

Now let us check the GitHub account.



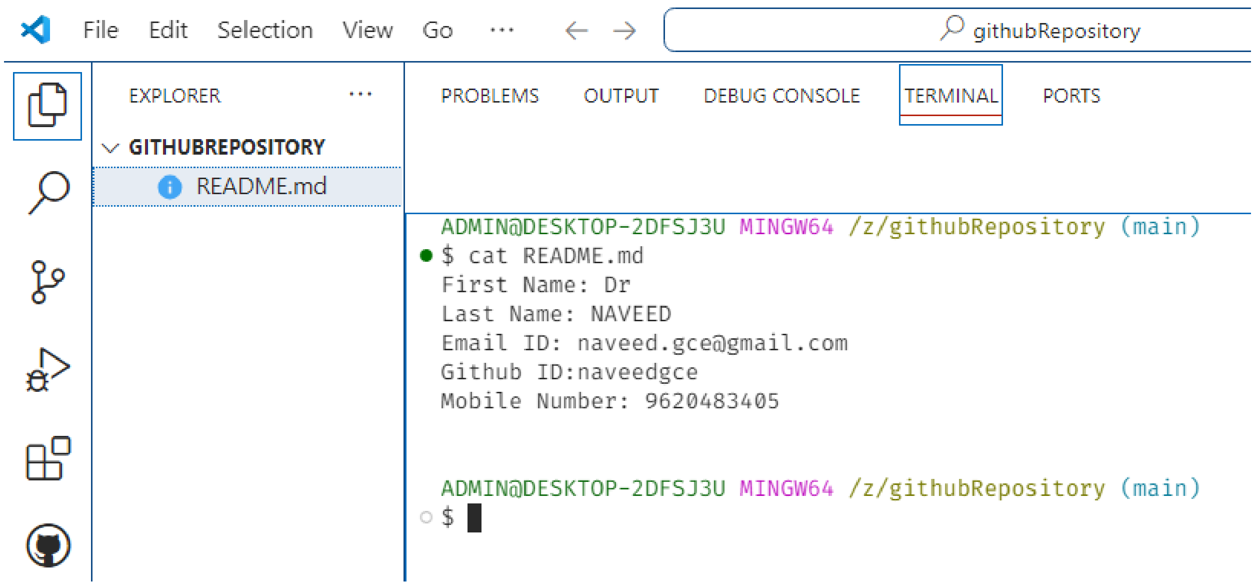
Dr.NAVEED Updated README.md file successfully 2fde9fd · 5 minutes ago 2 Commits

README.md Updated README.md file successfully 5 minutes ago

First Name: Dr Last Name: NAVEED Email ID: naveed.gce@gmail.com Github ID:naveedgce Mobile Number: 9620483405

We can see the information is updated. Earlier README.md file was empty. Now it shows the data.

OUTPUT:



```
ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/githubRepository (main)
$ cat README.md
First Name: Dr
Last Name: NAVEED
Email ID: naveed.gce@gmail.com
Github ID:naveedgce
Mobile Number: 9620483405


ADMIN@DESKTOP-2DFSJ3U MINGW64 /z/githubRepository (main)
$
```

VIVA-VOCE

1. **What is Git?**
   Git is a distributed version control system used to manage project code and track changes over time.
2. **Who developed Git and why?**
   Linus Torvalds developed Git in 2005 for managing Linux kernel development after the community stopped using BitKeeper.
3. **What are the three states in Git?**
   Modified, Staged, and Committed – representing a file's current workflow stage.
4. **What is a repository in Git?**
   A repository is a project directory tracked by Git, containing all version history and files.
5. **How do you initialize a Git repository?**
   By using the command git init inside the project folder.
6. **What is a staging area in Git?**
   A temporary area where changes are prepared (staged) before committing to the repository.
7. **What is a commit in Git?**
   A commit saves the current staged changes to the Git repository permanently with a message.
8. **What command is used to set user identity in Git?**
   git config --global user.name "Name" and git config --global user.email "email@example.com"
9. **What is the command to view current Git configuration?**
   git config --list
10. **How to clear the Git Bash screen?**
    Use clear or press Ctrl + L.

11. **What does git add . do?**
    It stages all the modified and new files in the current directory for the next commit.
12. **How to create a new file and add content?**
    Use nano filename or touch filename, then edit and save.
13. **What is the purpose of a README.md file?**
    It provides basic documentation and information about the repository.
14. **How to check the status of your working directory?**
    git status shows changes, untracked files, and staged files.
15. **How do you create a new branch?**
    git branch feature-branch creates a new branch named “feature-branch”.
16. **How to switch branches in Git?**
    Use git checkout branch-name.
17. **What is merging in Git?**
    Merging combines changes from one branch into another.
18. **How to merge a feature branch into master?**
    First switch to master using git checkout master, then use git merge feature-branch.
19. **What happens if you modify a file in a feature branch?**
    Changes remain local to that branch until merged into master.
20. **How do you delete a branch?**
    Use git branch -d branch-name.

21. **What are Git tags?**
    Tags mark specific points in Git history, often used for releases.
22. **Difference between lightweight and annotated tags?**
    Lightweight tags are simple pointers; annotated tags store metadata like author and date.
23. **How to create a lightweight tag?**
    git tag v1.0
24. **How to create an annotated tag?**
    git tag -a v2.0 -m "Tag message"
25. **How to list all tags in a repository?**
    Use git tag

26. **Can tags be pushed to remote repositories?**
    Yes, using git push origin tagname
27. **Can you delete a tag?**
    Yes, use git tag -d tagname
28. **How to view details of an annotated tag?**
    Use git show tagname
29. **Are tags part of branches?**
    No, tags are separate and point to commits, not branches.
30. **Why use tags in Git?**
    To mark release versions or important milestones in the codebase.

31. **What is git log used for?**
    To view the commit history with author, date, and message.
32. **How to see commits by a specific author?**
    git log --author="Author Name"
33. **How to view commit logs in brief?**
    git log --oneline
34. **How to view commits between dates?**
    git log --author="Name" --since="YYYY-MM-DD" --until="YYYY-MM-DD"
35. **How to view a specific commit using ID?**
    Use git show commit-ID
36. **What does git diff do?**
    Shows the difference between files or commits.
37. **Can Git log be used to see changes across branches?**
    Yes, using options like git log branch-name.
38. **How to view the last 5 commits?**
    git log -n 5
39. **Can we search commit messages?**
    Yes, with git log --grep="message"
40. **What is git blame used for?**
    To find who last modified a particular line of code.

41. **What is git cherry-pick?**
    It applies a specific commit from another branch into the current branch.
42. **Use case of cherry-pick?**
    Useful when you want only specific changes from a feature branch.
43. **How to undo a commit?**
    Use git revert commit-ID
44. **How to amend a previous commit message?**
    Use git commit --amend
45. **What is git reflog?**
    It records updates to the tip of branches, helping you recover lost commits.
46. **Can git cherry-pick cause conflicts?**
    Yes, conflicts may arise if the same content was modified.
47. **How to resolve cherry-pick conflicts?**
    Edit the file, resolve the conflict, then commit.
48. **How to change a commit message in history?**
    Use git rebase -i and modify messages carefully.
49. **How to check file history using Git?**
    Use git log filename
50. **What is git revert vs git reset?**
    git revert creates a new commit that undoes changes; git reset changes commit history.

51. **What is GitHub?**
    A platform for hosting Git repositories and collaborating on projects online.
52. **How to clone a GitHub repository to VS Code?**
    Use git clone URL and open it in VS Code.
53. **How to push local changes to GitHub?**
    Use git push origin branch-name
54. **What is git remote add origin?**
    It links the local repository to a remote GitHub repo.

55. **How to configure Git in VS Code?**
Set Git path in settings and enable Git extension.
56. **What is a pull request?**
A request to merge changes from one branch to another in GitHub.
57. **How to create a GitHub repository?**
Click on "New" in GitHub, name the repository, and set options.
58. **How to view commit history in GitHub?**
Navigate to the "Commits" tab in the repository.
59. **What is GitHub Actions?**
A CI/CD feature to automate workflows in GitHub.
60. **What are forks in GitHub?**
A personal copy of someone else's repository to contribute.

61. **How to rename a file in Git?**
git mv oldname newname
62. **How to remove a file from Git?**
git rm filename
63. **What is .gitignore?**
A file that specifies which files Git should ignore.
64. **How to list all branches?**
git branch
65. **What is HEAD in Git?**
It refers to the current commit your working directory is based on.
66. **What is a detached HEAD state?**
When HEAD points directly to a commit, not a branch.
67. **Can Git track empty directories?**
No, Git only tracks files.
68. **How to revert to a previous commit?**
Use git checkout commit-ID
69. **How to reset staging area?**
git reset unstages files.
70. **How to discard changes in working directory?**
git checkout -- filename

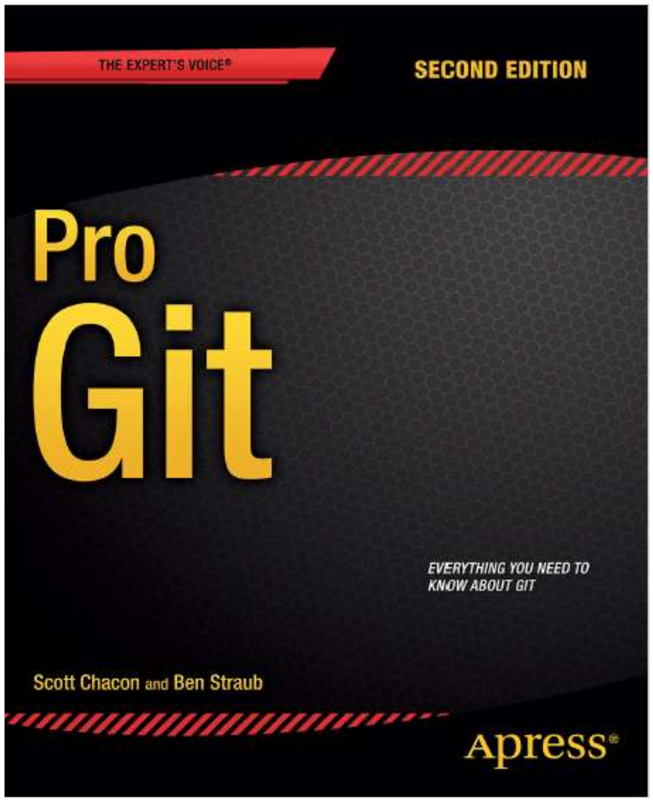
71. **Why use branches in projects?**
To work on features independently without affecting the main codebase.
72. **When should you use tags in a project?**
During stable releases or version marking.
73. **Why is Git preferred in large projects?**
Due to its distributed architecture, speed, and branching efficiency.
74. **What is the role of commits in collaboration?**
They provide checkpoints for tracking changes and debugging.
75. **How does Git help in team development?**
Enables parallel development, tracking contributions, and resolving conflicts.
76. **How often should you commit changes?**
Frequently, with meaningful commit messages.
77. **Why commit messages are important?**
They describe changes clearly for future reference.
78. **How can Git help in backups?**
Local and remote repositories serve as backups.
79. **Why use Git over traditional file copy methods?**
Git provides version control, history, and collaboration support.
80. **What is the advantage of using VS Code with Git?**
Integrated terminal, Git GUI, and ease of code editing.

81. **What is version control?**
A system for tracking changes in files over time.
82. **Difference between Git and GitHub?**
Git is the version control system; GitHub is a hosting platform for Git repositories.
83. **What is a commit hash?**
A unique ID for each commit, used to reference it.

84. **Why is Git distributed?**
    Each user has a complete copy of the repository.
85. **Can you work offline with Git?**
    Yes, most operations are local and can sync later.
86. **Why use git init?**
    To start tracking a new project with Git.
87. **Is Git case-sensitive?**
    Yes, file names in Git are case-sensitive on case-sensitive systems.
88. **What happens when two users edit the same file?**
    Git highlights conflicts during merge.
89. **What is a conflict in Git?**
    When Git cannot automatically resolve differences between changes.
90. **How to resolve conflicts?**
    Manually edit the file and commit resolved version.
91. **What is the use of .md files in GitHub?**
    Markdown files for documentation and project info.
92. **What command is used to edit files in terminal?**
    nano filename
93. **How to stage only specific files?**
    git add filename1 filename2
94. **What is the role of git push?**
    Uploads local commits to the remote repository.
95. **What command sets the default branch name in newer Git versions?**
    git config --global init.defaultBranch main
96. **What is the importance of lab record in Git course?**
    Essential for tracking learning progress, assessed during CIE.

Reference:

The theory part of the material from page number 8 to 16 is extracted from the Prescribed Textbook by the University as shown below.


THE EXPERT'S VOICE®
SECOND EDITION
Pro
Git
EVERYTHING YOU NEED TO KNOW ABOUT GIT
Scott Chacon and Ben Straub
Apress®

ASSIGNMENT

Q1.(EXP-01)BasicSetupandCreationofaNewRepository Aim:

1. Tocreateanewrepository“1WT23CS000”under anydiscdrivesuch asZ drive andalso a sub repository “aboutMyCollege”.
2. Tomakethe defaultbranch as masterbranchin some systemsit is main branch.
3. Tolaunchthegit andentertheuserconfigurationslikename,emailID.

(EXP-02)Toadd README.mdfileintotheRepository

Aim:

1. ToaddREADME.mdfileintotherepository/z/1WT23CS000/aboutMyCollegeunder master branch.
2. Toaddthe contentsgiven below:

```
Title: GITW
Full Name: Ghousia Institute of Technology for Women
College Code: WT
Affiliation: VTU, Belagavi
Year of Establishment: 2023
No. of Branches: 03
Departments: CS, IS and EC
Mobile No: 080-25536527
Email ID: principalgitw@gmail.com
Address: DRC post, near Dairy Circle Hosur Road Bangalore-
560029
```

3. Tocommitwithamessage“Contentsupdatedsuccessfully”

(Exp-03)Creating andManagingBranches

Aim:

1. Tocreateanewbranchnamed "feature-branch”.
2. To add a text file “aboutMyCollege.txt” into the repository /1WT23CS000/aboutMyCollege.
3. Toaddthe contentsinto thetext filegiven below:

```
Title: GITW
Full Name: Ghousia Institute of Technology for Women
College Code: WT
Affiliation: VTU, Belagavi
Year of Establishment: 2023
No. of Branches: 03
Departments: CS, IS and EC
Mobile No: 080-25536527
Email ID: principalgitw@gmail.com
Address: DRC post, near Dairy Circle Hosur Road Bangalore-
560029
```

4. Tocommitamessage“AddedtextfileaboutMyCollege.txt successfully”.

(Exp-04)MergingofFeature-BranchintotheMasterBranch: Aim:
Tomergefeature-branchintothemasterbranch
Experiment-05: UpdatingoffilesintheFeature-Branch
Aim:

1. ToaddtheBriefReviewofGITWinthefile“aboutMyCollege.txt”ofthefeature- branch:

GITW was established in the year 2023, affiliated with Visvesvaraya Technological University (VTU), Belagavi, Karnataka. Recognized by AICTE, New Delhi, and the Government of Karnataka, ranking first in women's minority education in the state. The college provides hostel facilities and organizes diverse programs enhancing students' overall personality. It offers B.E Programs in:

- ComputerScience&Engineering
- InformationScience&Engineering
- Electronics&CommunicationEngineering

Tocommit theabovewitha message“AddedbriefreviewofGITW Successfully”
Tomergethefeature-branchwiththemasterbranchandcommitwithamessage“Mergedthe feature-branch Successfully and updated the file with review Details”.

Q.5(Exp-06)LightWeightandAnnotatedTags Aim:

1. To update the file “README.md” and add “Date of Joining to GITW: 1st Oct-2023” to it under the repository /1WT23CS000/aboutMyCollege under master branch.
2. Tocommitwith amessage“My Date ofJoining to GITWUpdatedsuccessfully”
3. Toadd alight wightag v1.0 into file“README.md”
4. ToaddanAnnotatedtag v2.0withamessage“My DateofJoiningtoGITW”intothe same file

Q.6(Exp-09)GitinVSCode Aim:

1. Toclonetherepository/z/1WT23CS000/aboutMyCollege fromGit-BashtoVS code:
2. ToaddanewfilevsFile.txtunderVScode.Addthefollowingdata: Vs

File details:

- Version:2.1
- DateofInstallation: 30/07/2024
- AuthorName:Dr.Naveed

andcommitamessage “AddedvsFile.txt successfully”

SUMMARY OF CODING

EXPERIMENT-01

```
// To start the git and to make master branch as default branch
$ git init
// To register your name (author name) and Email address for the
   new project
$ git config --global user.name “Dr.NAVEED”
$ git config --global user.email naveed.gce@gmail.com
// To check the author name , email address and other data:
$ git config --list
// To check author name of the project
$ git config user.name
// To check email address of the author
$ git config user.email
// To clear the screen of the terminal (Ctrl+L)
$ clear
// T get into the Z drive of the computer hard disk. It can be
   any symbol
$ cd /z
// To create a repository “1WT23CS000”
$ mkdir 1WT23CS000
// To get back to the repository
$ cd 1WT23CS000
// To create another sub-repository “aboutMyself”
$ mkdir aboutMyself
// To get back to aboutMyself
$ cd aboutMyself
// To get back in single shot
$ cd /z/1WT23CS000/aboutMyself
```

EXPERIMENT-02

```
//To make any drive like z drive as a safe directory
$ git config --global --add safe.directory z:/
// To add a new file like README.md as untracked file
$ echo > README.md
// To track the file like README.md
$ git add .
//To commit with a message like Added README.md file successfully
$ git commit -m "committed message"
// To edit the file like README.md in an open screen
$ nano README.md
// To read the contents of a file like README.md under terminal
$ cat README.md
//To check the current status
$ git status
```

EXPERIMENT-03

```
//To check the current branch as well to check out number

  of branches available
$ git branch
// To create a new branch such as feature-branch
$ git branch feature-branch
// To shift the directory to the new branch such as feature-branch

$ git checkout feature-branch

// To shift back the directory to the default master branch

$ git checkout master

// To checkout number of files available in the repository

$ git ls-files
```

EXPERIMENT-04

```
// To merge feature-branch into master branch which will add all
   files of feature-branch into master branch
$ git merge feature-branch
```

EXPERIMENT-05

```
// To delete a branch such as feature-branch
$ git branch -d feature-branch
// To get reference code of deleted branch
$ git reflog
// To restore the deleted branch such as feature-branch with
   the given ref.code
$ git checkout -b feature-branch ref.code
(in updated version ref.code is not needed)
```

EXPERIMENT-06

```
// To add a light weight tag such as v1.0
$ git tag v1.0
// To check the number of tags available

$ git tag

// To check a particular tag such as v1.0

$ git show v1.0

// To add an annotated tag such as v2.0

$ git tag -a v2.0 -m “committed message”
```

EXPERIMENT-07

```
// To check the committed messages in master branch with full details
$ git log master

// To check the committed messages in master branch in brief with just one line
$ git log master --oneline
```

```
//To check committed message of a particular task with the given reference code ID

$ git show ref.code ID

// To check commits of a particular author from date to desired date with full details

$ git log --author="authorName" --since="YEAR-MONTH-DAY" until="YEAR-MONTH-DAY"

// To check commits of a particular author from date to desired date in brief with just one line

$ git log --author="authorName" --since="YEAR-MONTH-DAY" until="YEAR-MONTH-DAY" --oneline
// To check last 5 commits made in master branch with complete details
$ git log master -n 5
// To check last 5 commits made in master branch in brief with just one line.
$ git log master -n 5 --oneline
Note: Press Q to get back to the coding.
```

EXPERIMENT-08

```
// To check the committed messages in feature- branch
$ git log feature-branch --oneline
// To check committed message of feature-branch with ref.ID in master branch
$ git cherry-pick ref.ID
// To open the conflicting file such as aboutMyself.txt

$ vim aboutMyself.txt

//To delete a file such as data1.txt

$ git rm data1.txt

//To modify the committed message with Ref.ID

$ git revert Ref.ID

//To display the committed message with Ref.ID

$ git show Ref.ID
```

EXPERIMENT-09

```
// To get back to the current repository in Z drive such as 1WT23CS000/aboutMyself
$ cd /z/1WT23CS000/aboutMyself
// To clone the above repository in VS code after getting back to the current repository
$ code .
```

EXPERIMENT-10

```
// To clone repository in Z drive such as 1WT23CS000/aboutMyself into the GitHub account
$ git remote add origin URLcodeOfGitHubRepository
//To push the repository into the GitHub account if it is master branch
$ git push -u origin master
//To push the repository into the GitHub account if it is main branch
$ git push -u origin main
//To add a new branch such as feature-branch into GitHub account
$ git push -u origin feature-branch
```

GHOUSIA INSTITUTE OF TECHNOLOGY FOR WOMEN

Near Dairy Circle, Hosur Road, Bengaluru-560029, KARNATAKA

Affiliated to VTU., Belagavi, Recognized by Government of Karnataka & A.I.C.T.E., New Delhi

Contact

9986343109 / 9845954481
080 – 25536527

www.gitw.in

B.E Programs Offered

- Computer Science & Engineering
- Information Science & Engineering
- Electronics & Communication Engineering.

It was established in the year 2023, affiliated with Visvesvaraya Technological University (VTU), Belagavi, Karnataka. Recognized by AICTE, New Delhi, and the Government of Karnataka. It is one among the two engineering colleges for women in the state. The college provides hostel facilities and organizes diverse programs enhancing students' overall personality.