GIT

SCM (OR) Version Control

SCM is a process of tracking and controlling the changes in the software development. The popular SCM tools are listed below.

- Git
- SVN
- CVS
- Perforce
- ClearCase
- TFS (Team Foundation Service from Microsoft) ...etc.

SCM Goals

- Team work
- Defect tracking
- Configuration auditing
- Process management
- Environment management

SCM Terminologies

- Server/Client
- Repository
- Workspace
- Branch
- Check-In/Check-Out
- Revision
- Baseline

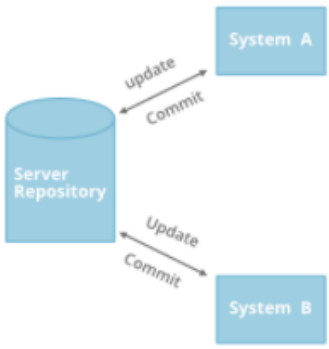
SCM Advantages

- Supports for non-linear deployment.
- It's fully distributed.
- Ability to handle large projects like Linux efficiently.
- Speed

Many of us aware about the version control when it comes to work with multiple developers on a single project and collaborate with them. There is no doubt that version control make developers work more easy and fast. In most of the organization developers use either Centralized Version Control System(CVCS) like Subversion(SVN) or Concurrent Version System(CVS) or Distributed Version Control System(DVCS) like Git (Written in C), Mercurial (Written in Python) or Bazaar (Written in Python).

SCM centralize model (CVCS)

In centralized source control, there is a server and a client. The server is the master repository which contains all of the versions of the code. To work on any project, firstly user or client needs to get the code from the master repository or server. So the client communicates with the server and pulls all the code or current version of the code from the server to their local machine. In other terms we can say, you need to take an update from the master repository and then you get the local copy of the code in your system. So once you get the latest version of the code, you start making your own changes in the code and after that, you simply need to commit those changes straight forward into the master repository. Committing a change simply means merging your own code into the master repository or making a new version off the source code. So everything is centralized in this model.



Centralized

Advantages of centralize model (CVCS)

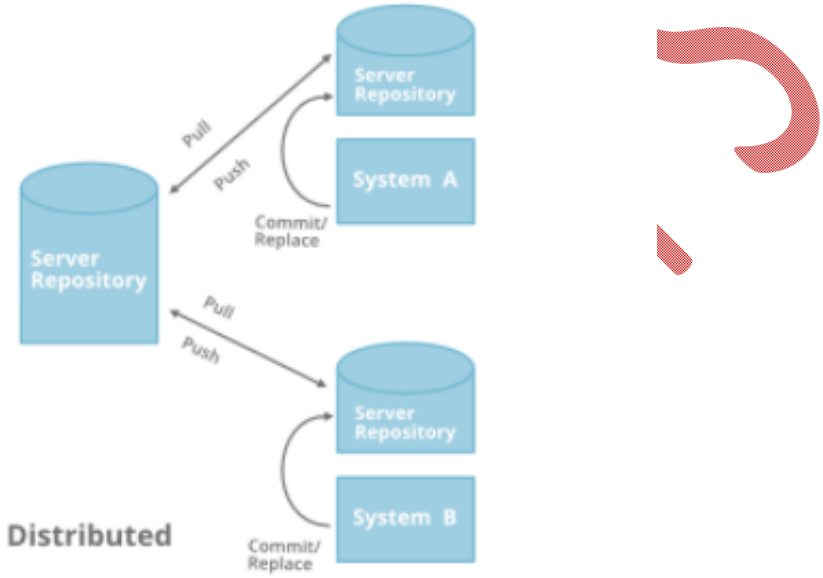
- Centralized version control is easier to learn than distributed model and easy to learn and easy to set up.
- If the project has long history or the project contain large binary files in that case downloading the entire project in DVCS can take more time and space than usual, whereas in CVCS you just need to get few lines of code because you don't need to save the entire history or complete project in your own server so there is no requirement for additional space.

Disadvantages of centralize model (CVCS)

- Network issue.
- Always connection between server and client.
- Multiple checkin/out for the different users.
- It requires lot of efforts and not friendly.
- If the main server goes down or it crashes in CVCS,We can't get back.
- It's slower than CVCS because need to communicate with the remote server for each and every command.
- Working on branches very difficult because it requires communicating with the server directly.
- Merge conflicts are more in CVCS in comparison of DVCS.

SCM distributed model (DVCS)

In distributed version control most of the mechanism or model applies the same as centralized. The only major difference you will find here is, instead of one single repository which is the server, here every single developer or client has their own server and they will have a copy of the entire history or version of the code and all of its branches in their local server or machine. Basically, every client or user can work locally and disconnected which is more convenient than centralized source control and that's why it is called distributed.



Advantages of distributed model (DVCS)

- In GIT there is no concept of server and client.
- In everything is workspace.
- Every workspace is a copy of repository.
- Its follows distributed model.
- It's faster than CVCS because you don't need to communicate with the remote server for each and every command.
- Working on branches is easy.
- Merge conflicts are less in DVCS in comparison of CVCS.

Disadvantages of distributed model (DVCS)

- Need to learn lot of commands to work on DVCS.
- If the project has long history or the project contain large binary files in that case downloading the entire project in DVCS can take more time compare to CVCS.

SVN Vs GIT

GIT	SVN
GIT It is a distributed version control system used for source code management.	Subversion (SVN) SVN is centralized versioning and revision control system.
GIT was developed for Linux kernel by Linus Torvalds.	SVN was developed by CollabNet, Inc.
Every user can maintain their own copy of code on their local similar to branch.	Everyone has a working copy and changes are made to a central repository.
It works on two stages commit and push.	It works on single stage i.e. commit
It Require very less Storage.	It Require much more storage compared to GIT.
It automatically remembers the revision of merges.	In SVN we need to remember revision about the last revision of merged.
It maintains information about the user who has performed the merge	All changes are done on branch appear to be made by merging user.
User can easily switch between different branches on same code base	To Work on branch user must copy the trunk into another directory and then merge it back when complete.
We can view merge specific changes.	Not possible to see merge related changes.
It can work in both online as well as offline mode, so network access is not mandatory.	Must be connected to network for commit.
It has easy to fork, branch and merge	Branching and merging is time consuming.
Used by **90%** of professional developers.	Used by **10%** of professional developers.
A sub project is called a Git “submodule”.	A sub project is called an “SVN external”.
GIT doesn’t have a global revision number.	SVN have a global revision number.
The content in Git is stored as metadata.	SVN stores files of content.
GIT contents are cryptographically check-summed using the SHA-1 hash algorithm.	SVN doesn’t have hashed contents.

What is Git?

GIT is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files.

Why Git is distributed model?

One developer can share the changes to the repository (OR) Other developers so it’s called distributed model.

- If someone works on project, They have to clone their local machines.In this case local machines behave like GIT server.
- The idea is everyone work locally and all operations are going very fast.

Why GIT is required in Projects?

- Developers easily collaborate and work on same project.
- Merging code changes are very easy.
- SCM tools records all versions with
  - Time stamp (Date && Time)
  - Developer name/mail
  - Commit message
- Debugging defects are simplified.
- If latest software release has any issue. we can easily rollback to its previous version.
- GIT is fast when it is compared with other version controlling tools.
- It also works as backing up our project code.

Workspaces in GIT

A workspace is simply a set of git repositories. There are 2-types workspaces in GIT.

1. Central workspace (Bare)
2. Local workspace/User workspace (Non Bare)

Central workspace (Bare)

It contains store and share the changes.

Ex:

- Github
- Bitbucket ...etc.

Local workspace/User workspace (Non Bare)

We can modify the changes store and share the changes.

What is GitHub?

GitHub is a Git repository hosting service. GitHub also facilitates with many of its features, such as access control and collaboration. It provides a Web-based graphical interface.

GitHub is an American company. It hosts source code of your project in the form of different programming languages and keeps track of the various changes made by programmers.

It offers both distributed version control and source code management (SCM) functionality of Git. It also facilitates with some collaboration features such as bug tracking, feature requests, task management for every project.

Features of GitHub

GitHub is a place where programmers and designers work together. They collaborate, contribute, and fix bugs together. It hosts plenty of open source projects and codes of various programming languages.

Some of its significant features are as follows.

- Collaboration
- Integrated issue and bug tracking
- Graphical representation of branches
- Git repositories hosting
- Project management
- Team management
- Code hosting
- Track and assign tasks
- Conversations

Benefits of GitHub

GitHub can be separated as the Git and the Hub. GitHub service includes access controls as well as collaboration features like task management, repository hosting, and team management.

The key benefits of GitHub are as follows.

- It is easy to contribute to open source projects via GitHub.
- It helps to create an excellent document.
- You can attract recruiter by showing off your work. If you have a profile on GitHub, you will have a higher chance of being recruited.
- It allows your work to get out there in front of the public.
- You can track changes in your code across versions.

GIT Vs GitHub

GIT	GitHub
Git is a distributed version control tool that can manage a programmer's source code history.	GitHub is a cloud-based tool developed around the Git tool.
A developer installs Git tool locally.	GitHub is an online service to store code and push from the computer running the Git tool.
Git focused on version control and code sharing.	GitHub focused on centralized source code hosting.
It is a command-line tool.	It is administered through the web.
It facilitates with a desktop interface called Git Gui.	It also facilitates with a desktop interface called GitHub Gui.
Git does not provide any user management feature.	GitHub has a built-in user management feature.
It has minimal tool configuration feature.	It has a market place for tool configuration.

Bitbucket

Bitbucket is a web-based version control repository hosting service owned by Atlassian, for source code and development projects that use either Mercurial or GIT revision control systems. Bitbucket offers both commercial plans and free accounts.

Features of Bitbucket

- The code review and codes are managed by Bitbucket which possesses many features such as continuous delivery lines, pull services for comments and code review, bitbucket pipelines, dual-step verification process, whitelisting of IP, merging and code searching of alphas and checks.
- It provides Git large file storage services, tracking of issues, Wikis, integrations, and add-ons.
- The documentation also includes readme files that have different markdown file formats.
- The static websites hosted on Bitbucket cloud servers have Bitbucket.io.domain.in the URL.
- The snippet and smart monitoring enable the developer to exchange the code files or segments and utilizes third-party servers that rely on any development and programming language. They supports for freestyle of Jenkins, project based on pipelines, and multiple branch pipeline is provided by Bitbucket.
- The automated webhook development in Bitbucket server repo if Jenkin jobs are saved.
- The rapid section of projects on the server using Jenkin's job via the dropdown.
- The Mercurial VCS and Git are supported by Bitbucket which is scripted in Python on the web framework of Django.
- It can be installed in most of the operating system such as Mac, Windows, and Android applications.
- Easily integrating with other Atlassian tools like JIRA, Confluence and CROWD ...etc.

GIT Client

For interaction between our local machine and remote git server(Github/Bitbucket) we need git client.GIT supports both GUI (Graphical User Interface) & CLI (Command Line Interface).

Ex:

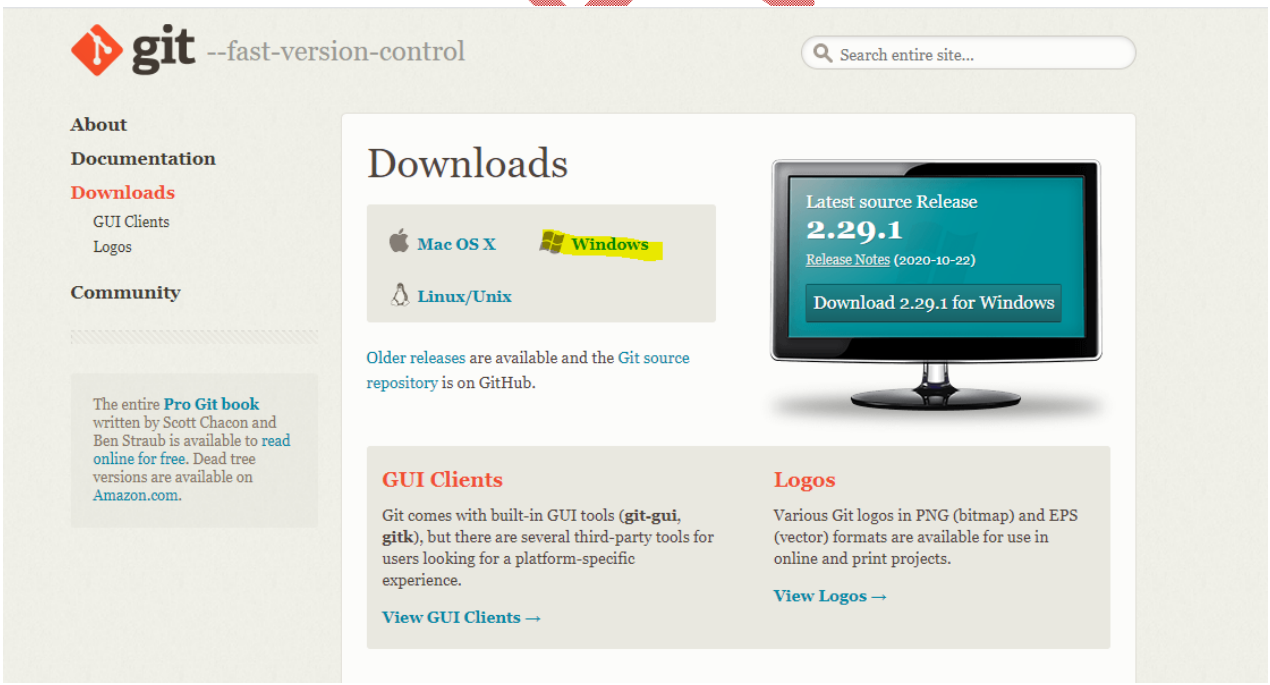
- Git bash
- Source tree
- Tortoisegit
- Git extension
- Smartgit
- Atom ....etc.

GIT Bash installation on Windows

To use Git, you have to install it on your computer. Even if you have already installed Git, it's probably a good idea to upgrade it to the latest version. You can either install it as a package or via another installer or download it from its official site.

Step-1

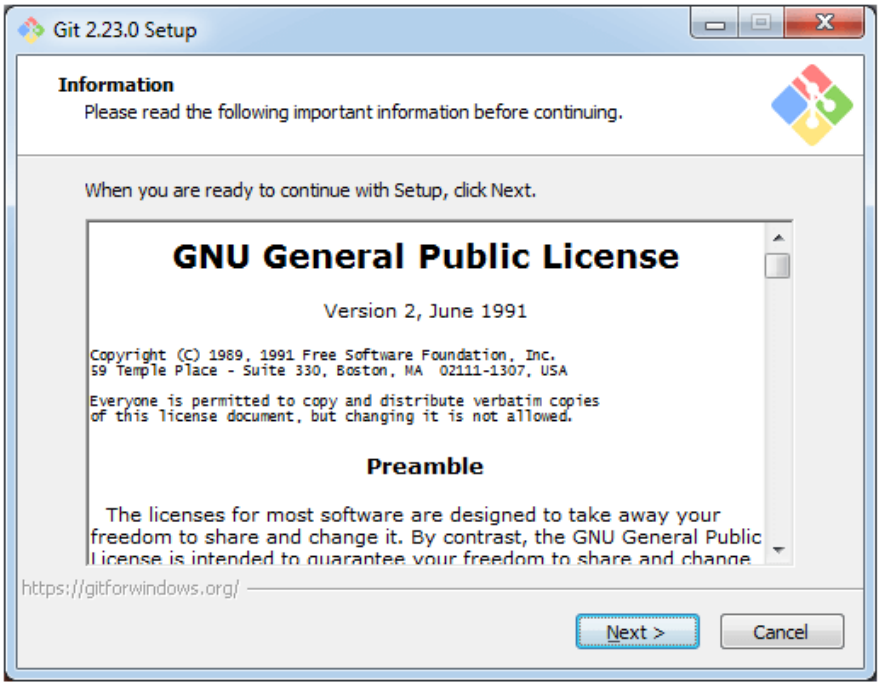
To download the Git installer, visit the Git's official site and go to download page. The link for the download page is https://git-scm.com/downloads.



Click on the package given on the page as download 2.23.0 for windows. The download will start after selecting the package. Now, the Git installer package has been downloaded.

Step-2

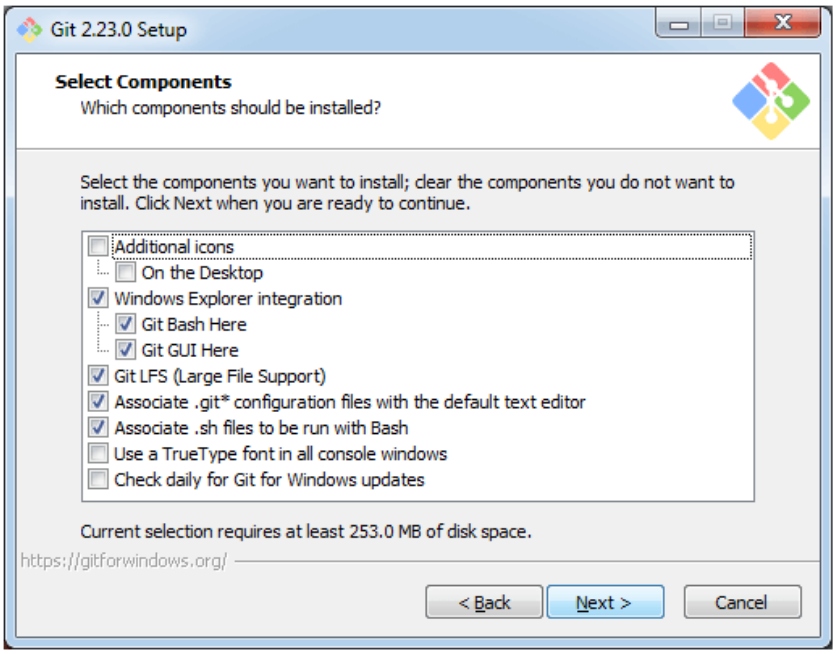
Click on the downloaded installer file and select yes to continue. After the selecting yes the installation begins, and the screen will look like as



Click on next to continue.

Step-3

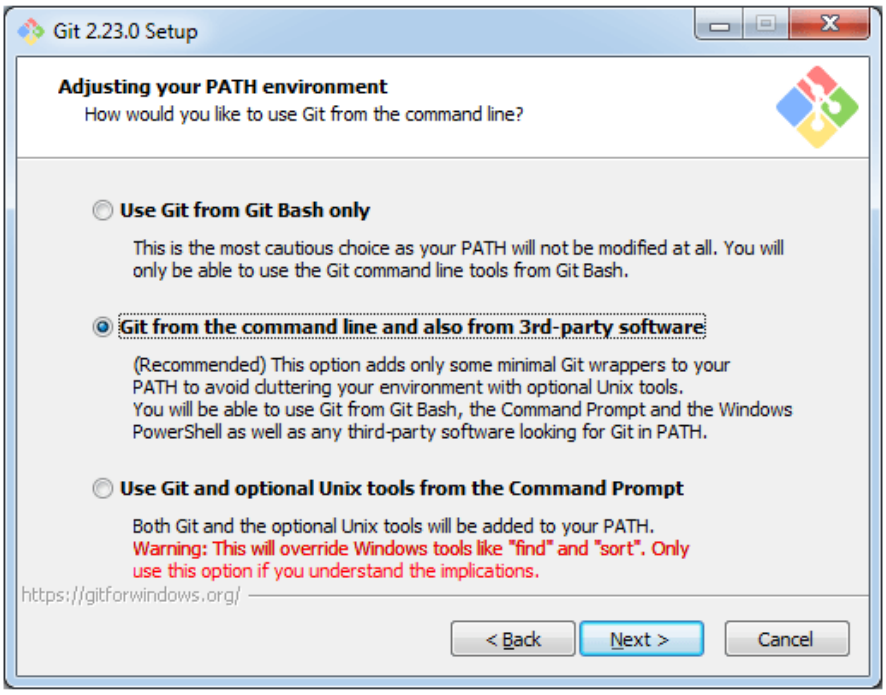
Default components are automatically selected in this step. You can also choose your required part.



Click next to continue.

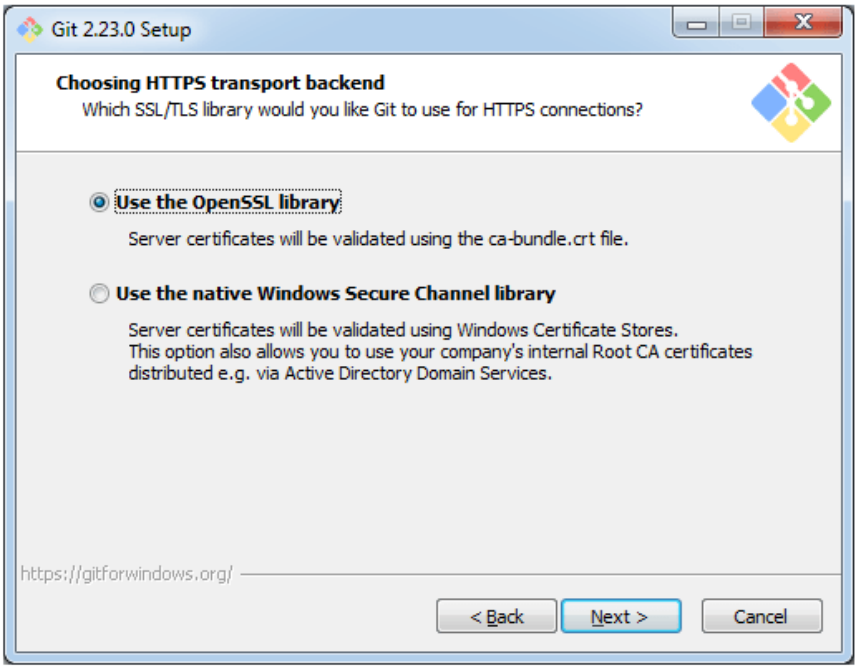
Step-4

The default Git command-line options are selected automatically. You can choose your preferred choice. Click next to continue.



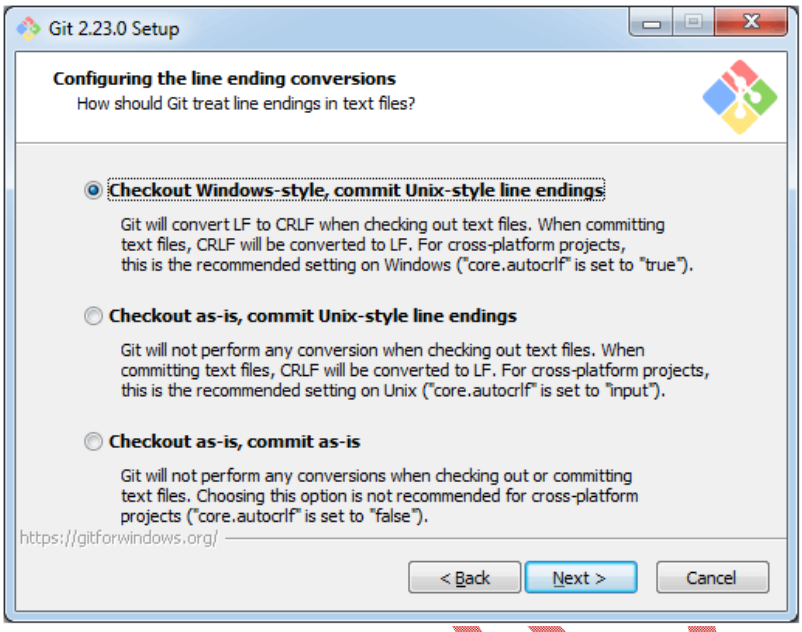
Step-5

The default transport backend options are selected in this step. Click next to continue.



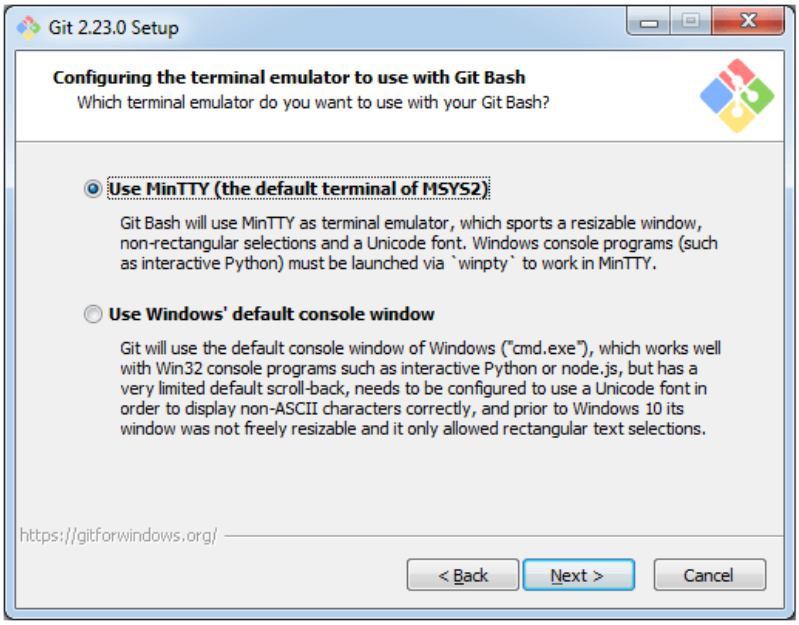
Step-6

Select your required line ending option and click next to continue.



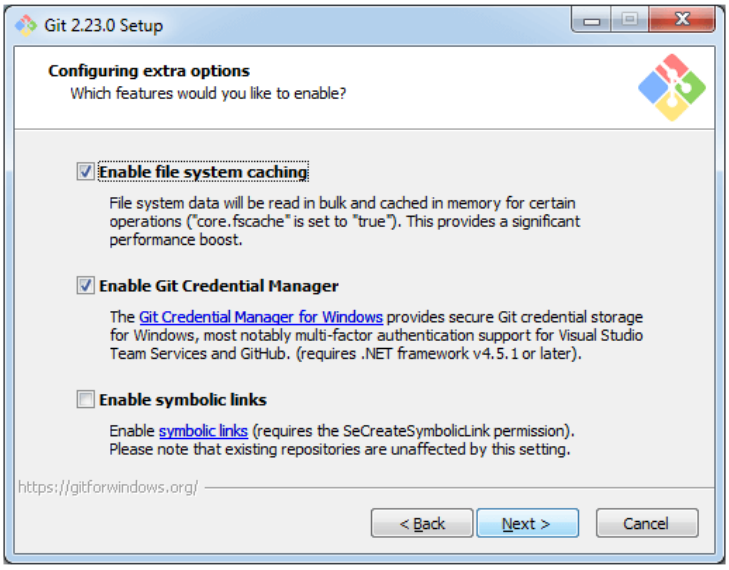
Step-7

Select preferred terminal emulator clicks on the next to continue.



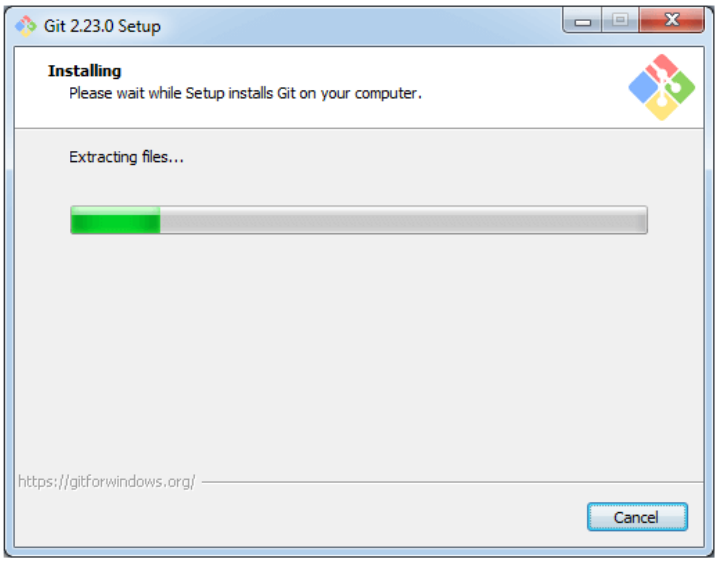
Step-8

This is the last step that provides some extra features like system caching, credential management and symbolic link. Select the required features and click on the next option.



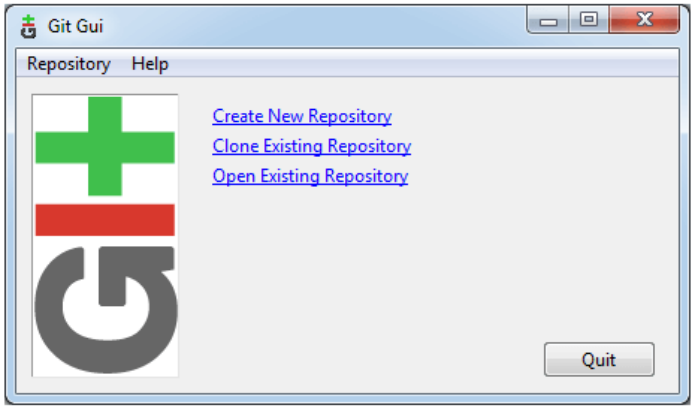
Step-9

The files are being extracted in this step.



Therefore, The Git installation is completed. Now you can access the Git Gui and Git Bash.

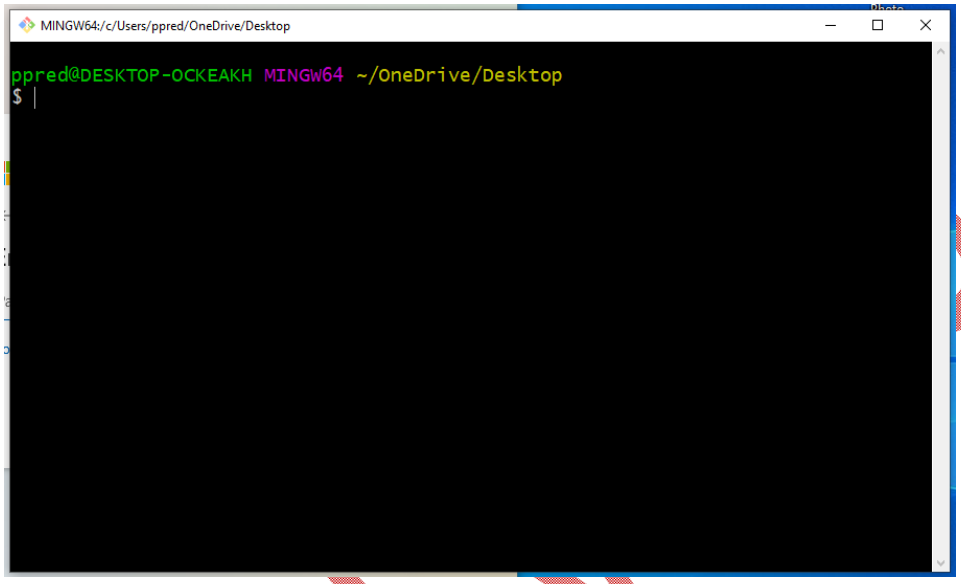
The Git Gui looks like as



It facilitates with three features.

- Create New Repository
- Clone Existing Repository
- Open Existing Repository

The Git Bash looks like as



GIT Client installation on Linux

Git is an open-source distributed version control system that is available for everyone at zero cost. It is designed to handle minor to major projects with speed and efficiency. It is developed to co-ordinate the work among programmers. The version control allows you to track and work together with your team members at the same workspace.

Step-1

Start the General OS and Package update.

yum update –y

```
[root@samplegoogle ec2-user]# yum update -y
Red Hat Update Infrastructure 3 Client Configuration Server 8                17 kB/s | 2.1 kB     00:00
Red Hat Enterprise Linux 8 for x86_64 - AppStream from RHUI (RPMs)           17 kB/s | 2.8 kB     00:00
Red Hat Enterprise Linux 8 for x86_64 - BaseOS from RHUI (RPMs)              16 kB/s | 2.4 kB     00:00
Dependencies resolved.
Nothing to do.
Complete!
[root@samplegoogle ec2-user]#
```

Step-2

To install Git, run the below command

yum install git* -y

```
[root@samplegoogle ec2-user]# yum install git* -y
Last metadata expiration check: 0:01:23 ago on Sun 25 Oct 2020 12:50:49 PM UTC.
Dependencies resolved.
================================================================================
 Package                 Arch       Version                                 Repository                  Size
================================================================================
Installing:
 git                     x86_64     2.18.4-2.el8_2                          rhel-8-appstream-rhui-rpms  187 k
 git-all                 noarch     2.18.4-2.el8_2                          rhel-8-appstream-rhui-rpms   48 k
 git-clang-format        x86_64     9.0.1-2.module+el8.2.0+5494+7b8075cf    rhel-8-appstream-rhui-rpms   20 k
 git-core                x86_64     2.18.4-2.el8_2                          rhel-8-appstream-rhui-rpms  4.0 M
 git-core-doc            noarch     2.18.4-2.el8_2                          rhel-8-appstream-rhui-rpms  2.3 M
 git-daemon              x86_64     2.18.4-2.el8_2                          rhel-8-appstream-rhui-rpms  708 k
 git-email               noarch     2.18.4-2.el8_2                          rhel-8-appstream-rhui-rpms   88 k
 git-gui                 noarch     2.18.4-2.el8_2                          rhel-8-appstream-rhui-rpms  296 k
 git-instaweb            x86_64     2.18.4-2.el8_2                          rhel-8-appstream-rhui-rpms   62 k
 git-subtree             x86_64     2.18.4-2.el8_2                          rhel-8-appstream-rhui-rpms   70 k
 git-svn                 x86_64     2.18.4-2.el8_2                          rhel-8-appstream-rhui-rpms  755 k
 gitk                    noarch     2.18.4-2.el8_2                          rhel-8-appstream-rhui-rpms  201 k
```

Step-3

When the central installation done, first check to ensure the executable file is set up and accessible. The best way to do this is the git version command.

git --version

```
[root@samplegoogle ec2-user]# git --version
git version 2.18.4
[root@samplegoogle ec2-user]#
```

GIT Client Configuration

This information is used by git to record our commits

git config --global user.name "psddevops"

git config --global user.email ppreddy2711@gmail.com

```
[ec2-user@samplegoogle ~]$ git config --global user.name "psddevops"
[ec2-user@samplegoogle ~]$ git config --global user.email "ppreddy2711@gmail.com"
[ec2-user@samplegoogle ~]$ git config --global push.default simple
[ec2-user@samplegoogle ~]$
```

--global tells git to use the same information for all the repositories i manage on my local machine.

git config --list

```
[ec2-user@samplegoogle ~]$ git config --list
user.name=psddevops
user.email=ppreddy2711@gmail.com
push.default=simple
[ec2-user@samplegoogle ~]$
```

git config --global --edit

```
[ec2-user@samplegoogle ~]$ git config --global --edit
[ec2-user@samplegoogle ~]$
```

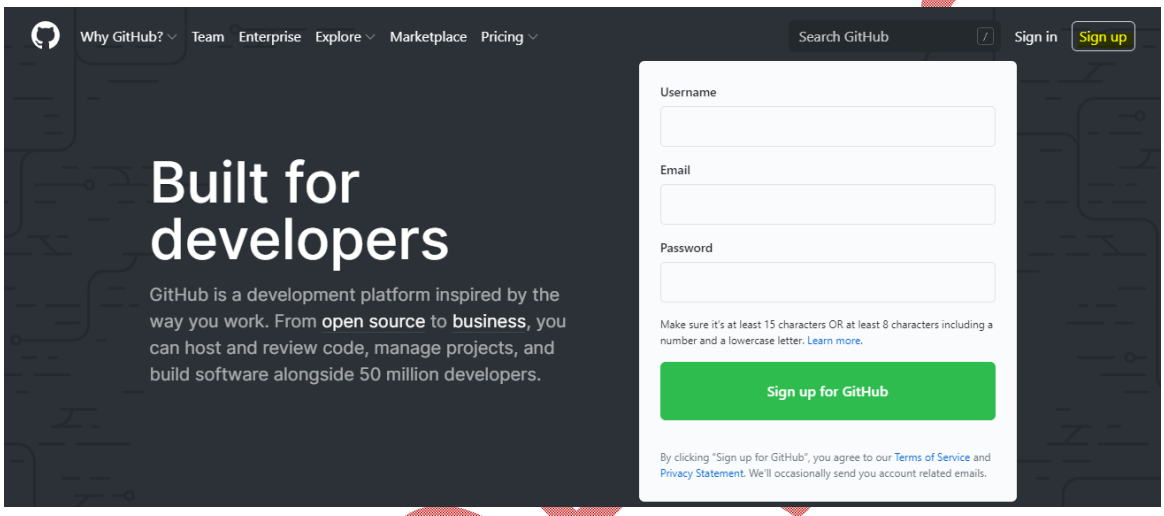
Checking GIT version

git --version

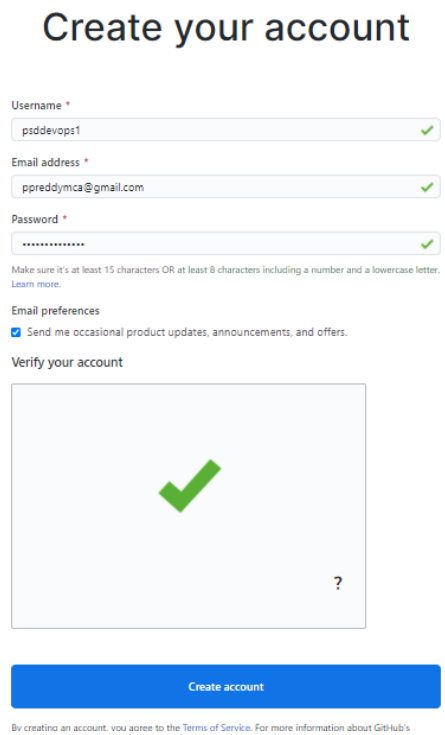
```
ppred@DESKTOP-OCKEAKH MINGW64 ~/OneDrive/Desktop
$ git --version
git version 2.29.1.windows.1
```

Signup to the Github

Login in to https://github.com/

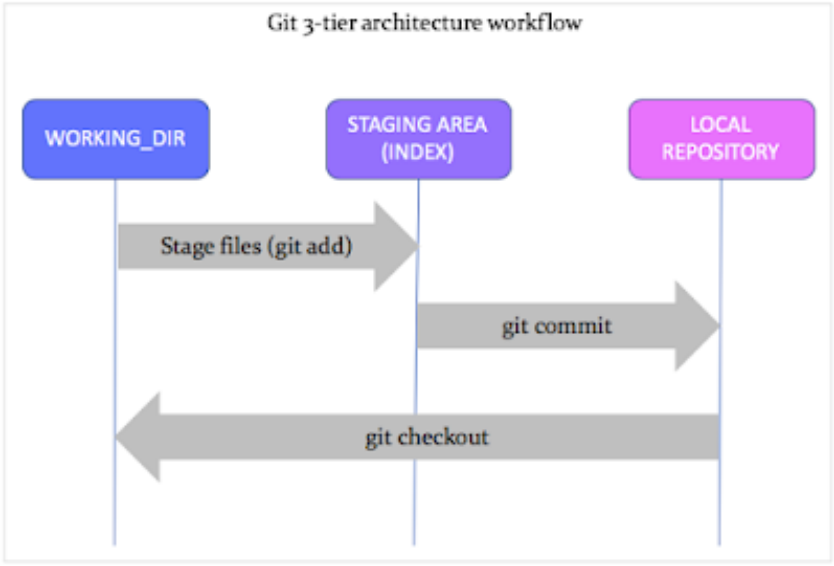


Click on to *Sign up* button and enter username, email address, password, verify and create the account.



*Prepared by **PPNREDDY** .....*

GIT Lifecycle



- Untracked
- Unmodified/Modified
- Staged

Working Directory (OR) Working Area

Any modifications we do to the local repository those modification are kept under working area.

Staging area/Index area/Buffer area

This area is to stage the files we want to commit to local & remote repository. Whichever files we have to commit those files we have to move staging area. In this area *commit-id* and *snap-shots* will be created.

Ex:

- git add README.md
- git add *
- git add *.java
- git add *.sh

Commit-id

- It's a 40-character random generated unique number.
- It's describing the file changes. Who did the change and when did the changes for same.

Snap-shot

What you have add and what you have modified the file differences is called snap-shot.

Local Repository/Repository

A repository contains a directory named .git, where git keeps all of its metadata for the repository.

git status

This provides information about our local repository (working area, stating area & local repository)

Commit:

When we do commit, it picks the files present in staging and commits to local repository. Its created commit-id contains username, mail, date and time other details.

Ex:

git commit -m 'Learning git tool'

Creating Central repository (Without Github/Bitbucket) and Local repository in Local machine

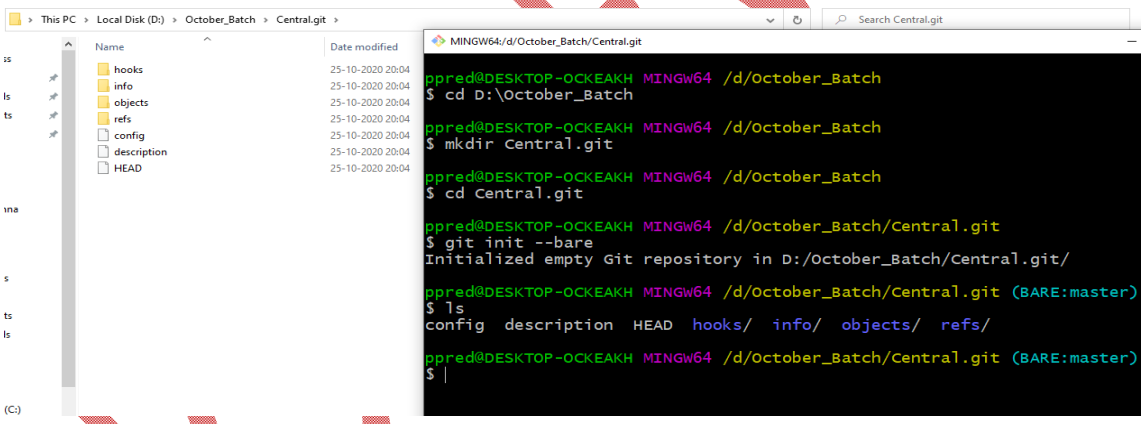
Creating bare/central repository

cd D:\October_Batch

mkdir Central.git

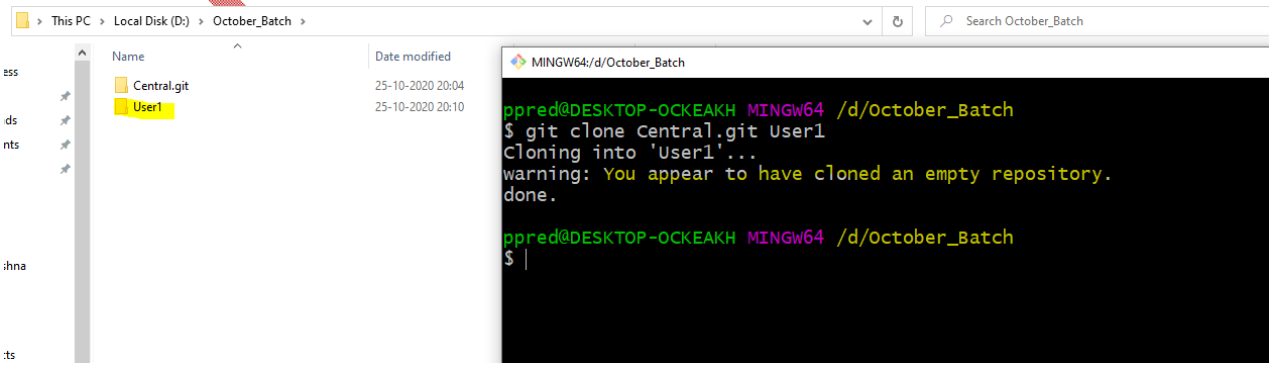
cd Central.git

git init --bare



Creating local repository/non-bare repository

git clone Central.git User1



git clone

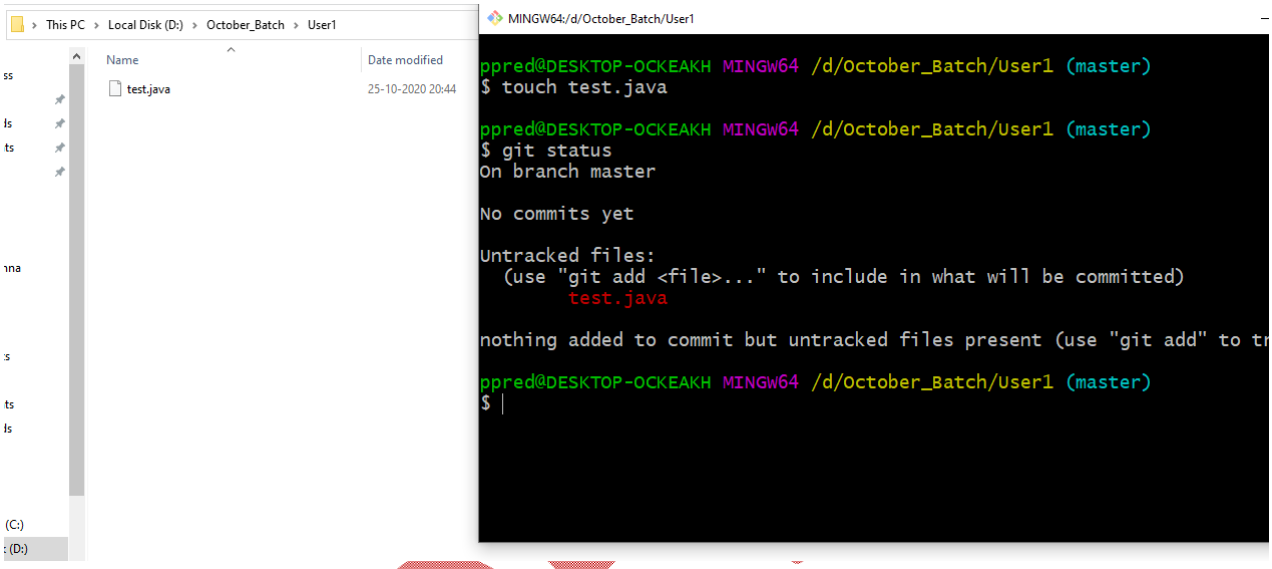
Git clone clones the remote copy into our local machine.

Adding and pushing a single file to the remote repo

Creating/Adding a file in working area

#touch test.java

#git status

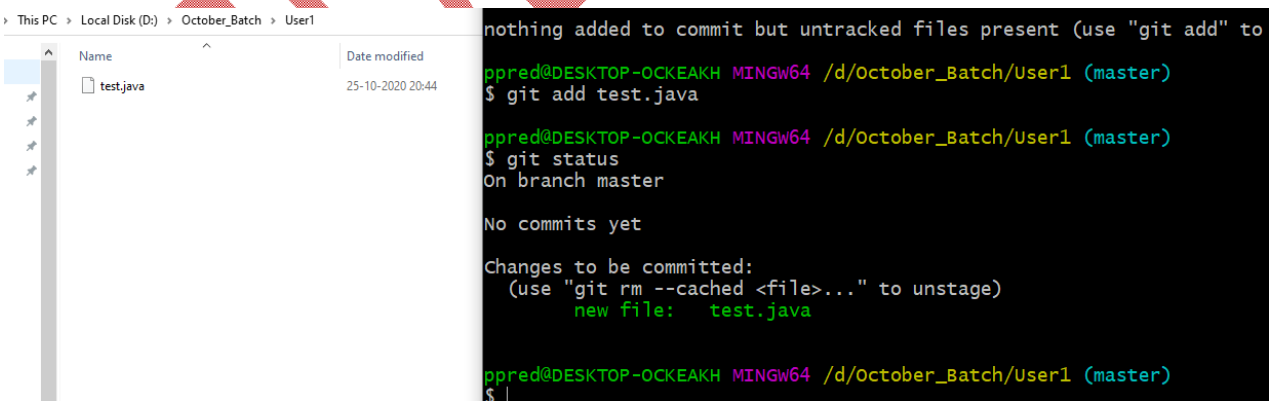


Untracked file

Untracked file is a file there is no previous information of that file i.e. new file.

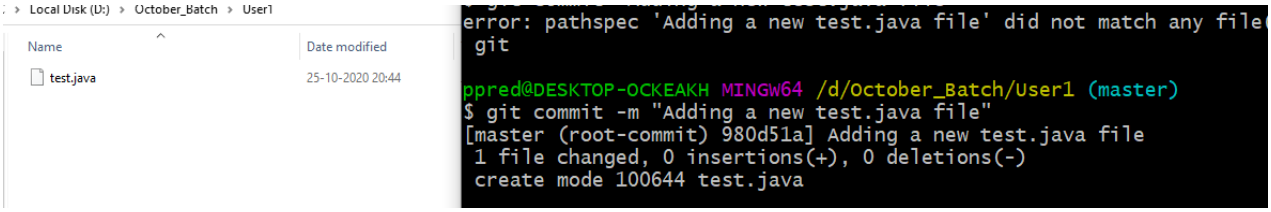
Moving a file working area to staging area

#git add test.java



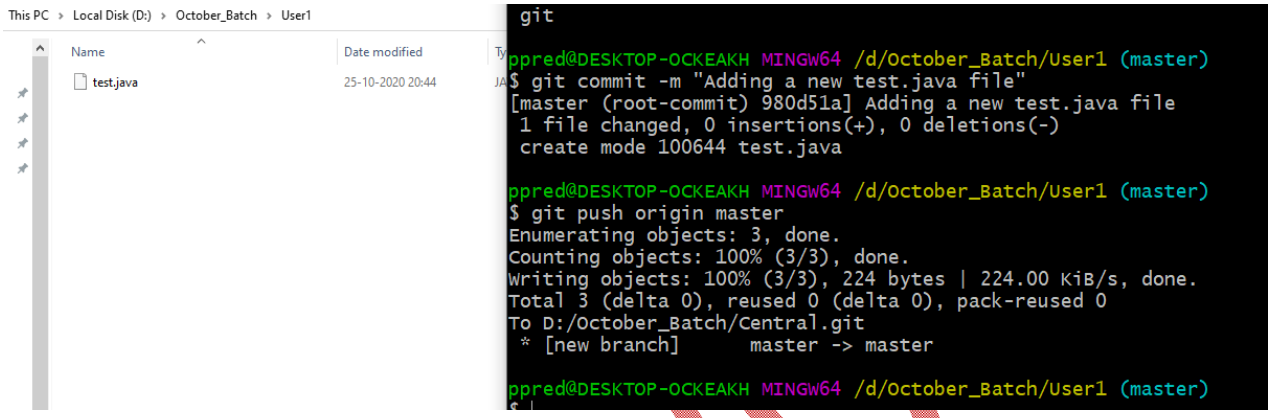
Pushing a file staging area to local repository

git commit -m "Adding a new test.java file"



Pushing changes to remote repo

git push origin master



Adding few more files in User1

#touch test1.java test2.java test3.java → Presented in working area

git add . → Presented in working area

git commit -m "Added three files" → added the files to the local repository

git push → Push to the remote repository

How can view the log files in GIT

By using *git log* command we can view the log information.It contains the information like Author,Date and Comment information.

#git log

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/User1 (master)
$ git log
commit ec5d2b877a18214afd768328caaedb3be9dafca7 (HEAD -> master
Author: psddevops <ppreddy2711@gmail.com>
Date:   Mon Oct 26 10:56:58 2020 +0530

    Added test4.java file

commit a563169802b95646da86ae826aec11c41b4923b2
Author: psddevops <ppreddy2711@gmail.com>
Date:   Mon Oct 26 10:46:37 2020 +0530

    Added three files

commit 980d51a63929bbdd3ac1387b8f4dd79af9748897
Author: psddevops <ppreddy2711@gmail.com>
Date:   Sun Oct 25 20:52:36 2020 +0530

    Adding a new test.java file

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/User1 (master)
$
```

git log --oneline

Display the log information in single line.

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/User1 (master)
$ git log --oneline
ec5d2b8 (HEAD -> master, origin/master) Added test4.java file
a563169 Added three files
980d51a Adding a new test.java file

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/User1 (master)
$
```

git log -<No.of commits>

git log -2 (Display latest two commits)

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/User1 (master)
$ git log -2
commit ec5d2b877a18214afd768328caaedb3be9dafca7 (HEAD -> master, origin/master)
Author: psddevops <ppreddy2711@gmail.com>
Date:   Mon Oct 26 10:56:58 2020 +0530

    Added test4.java file

commit a563169802b95646da86ae826aec11c41b4923b2
Author: psddevops <ppreddy2711@gmail.com>
Date:   Mon Oct 26 10:46:37 2020 +0530

    Added three files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/User1 (master)
$
```

git log --oneline -2 (Display latest two commits in one line)

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/User1 (master)
$ git log --oneline -2
ec5d2b8 (HEAD -> master, origin/master) Added test4.java file
a563169 Added three files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/User1 (master)
$
```

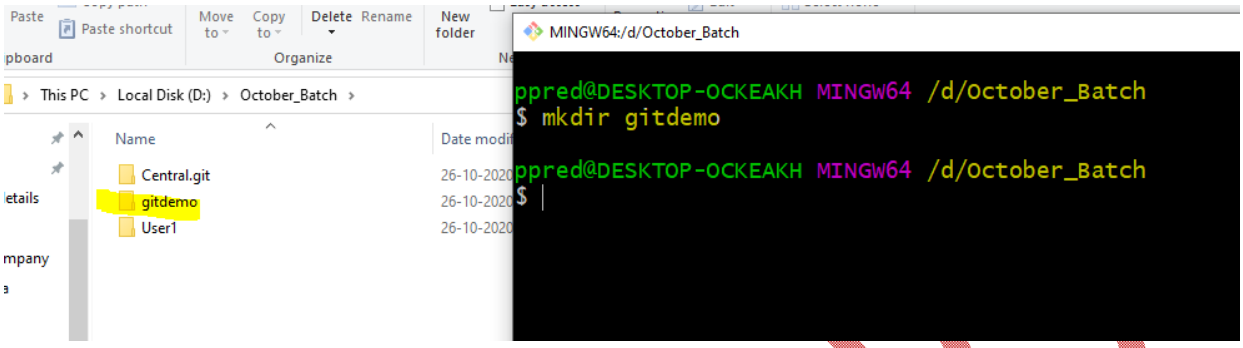
git log --oneline --all (Display all commits in one line)

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline --all
d027b7e (HEAD -> master) seven.java
55894f7 Added six.java
583b2a0 Added five.java
21bdc50 (origin/master) Added four.java file
0d9003d Added three.java
20bce21 Added the files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Create a project and Push to the remote repository (Git HUB)

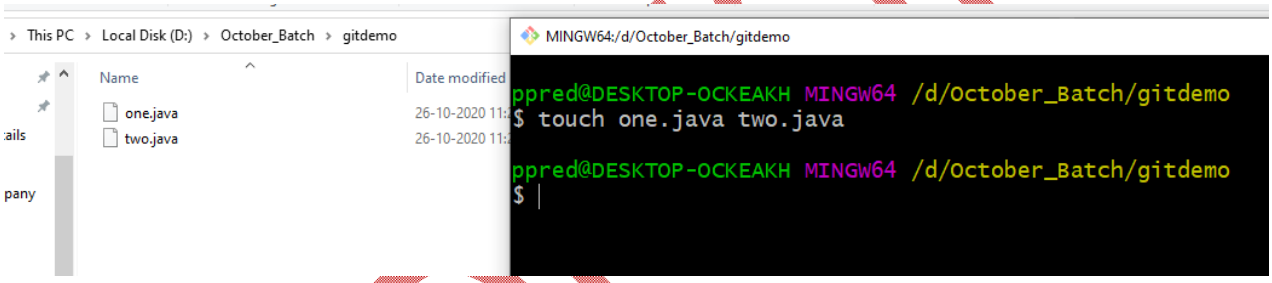
Create a folder “gitdemo” in local machine.

#mkdir gitdemo

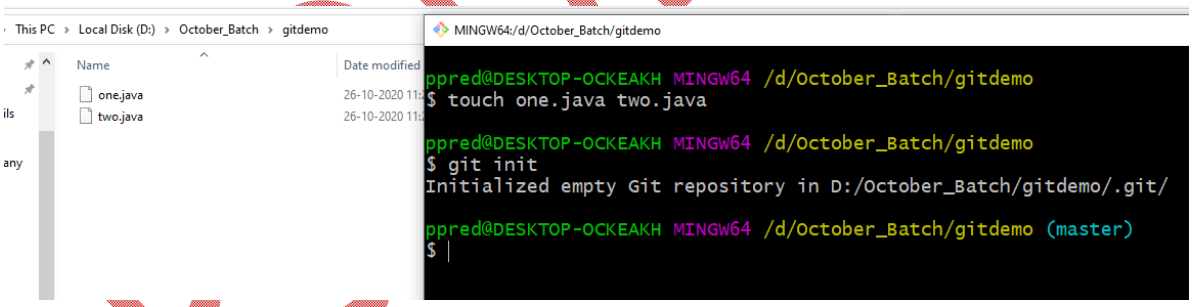


Add the files inside *gitdemo* folder

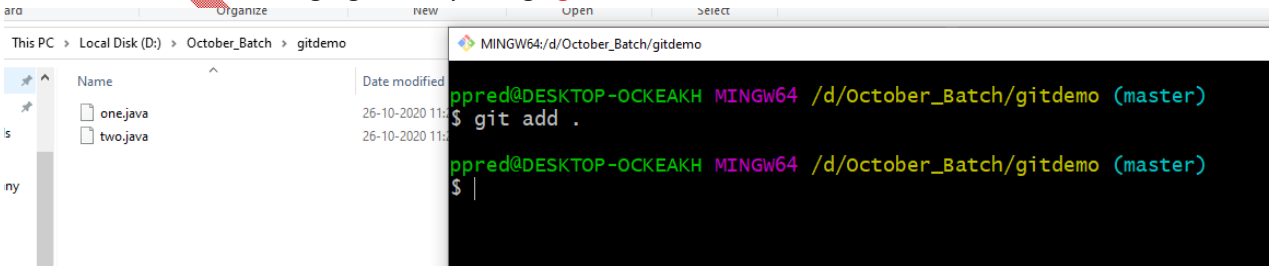
#touch one.java two.java



Initialize the git by using “git init” command.

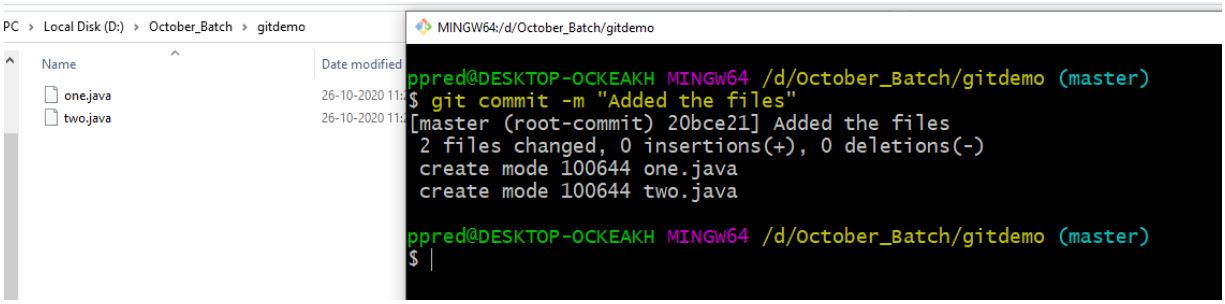


Move the files to the staging area by using “git add .” command.



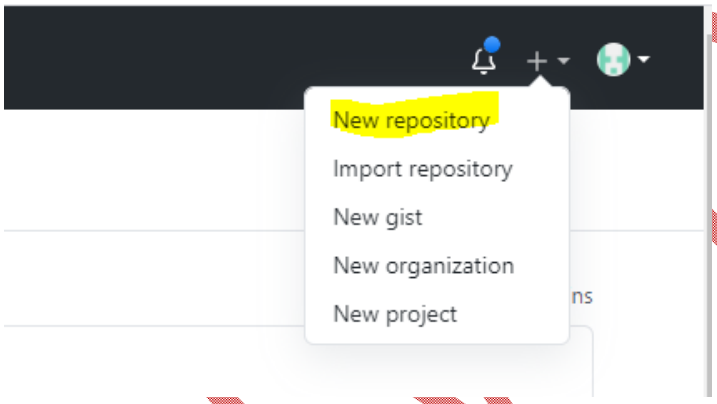
Move the files to the local repository by using “git commit -m “===” “command.

*Prepared by **PPNREDDY** .....*

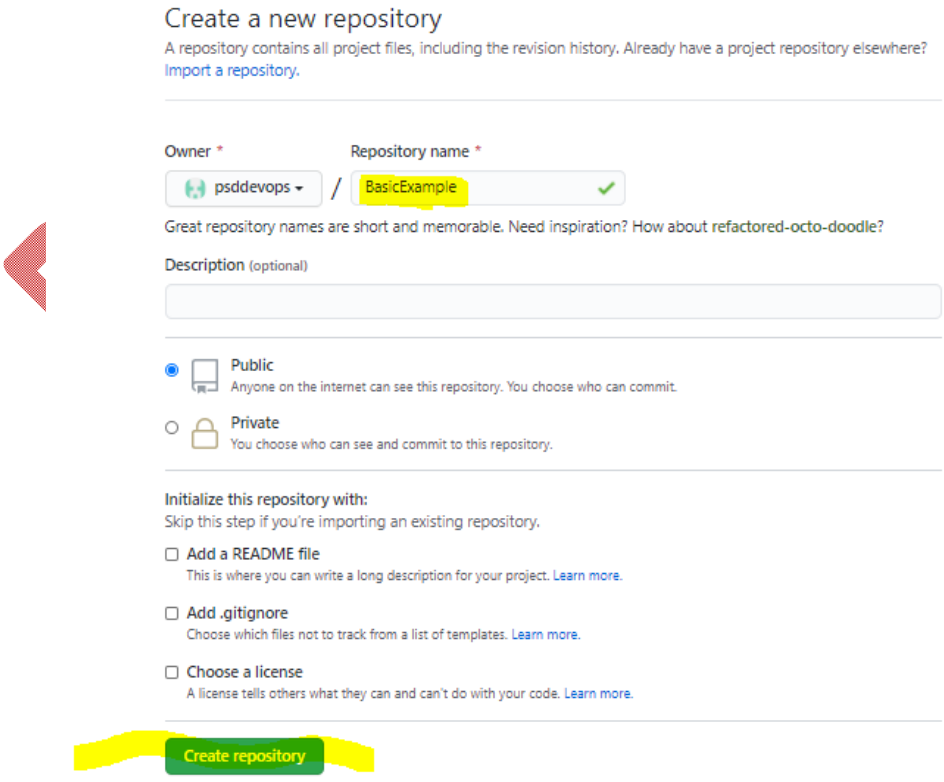


Login and create the new repository in github (https://github.com/)

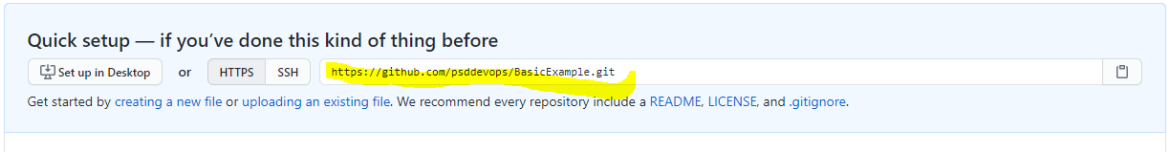
Click on (+) symbol on top right hand side and select new repository



Enter repository name "BasicExample" and create repository.

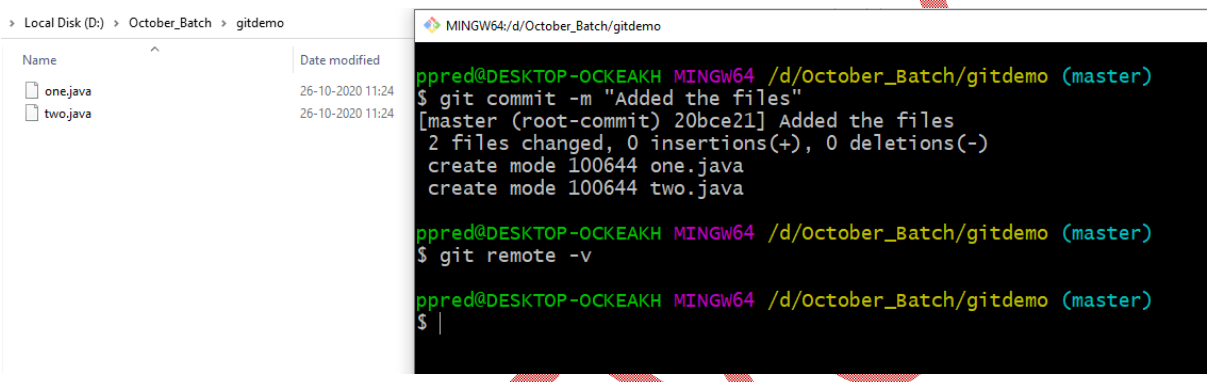


Copy the newly created URL (https://github.com/psddevops/BasicExample.git)



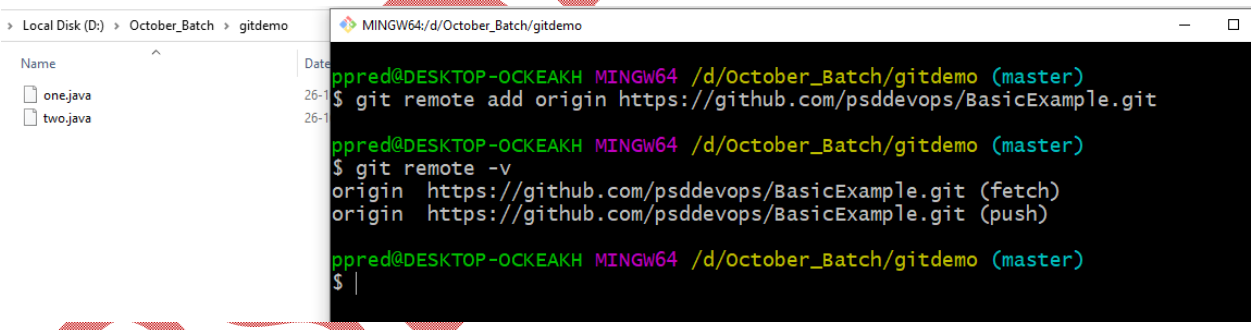
Verify the remote connection by using "git remote -v" command in git bash

#git remote –v



Adding the URL for the remote repository where your local repository code will be pushed.

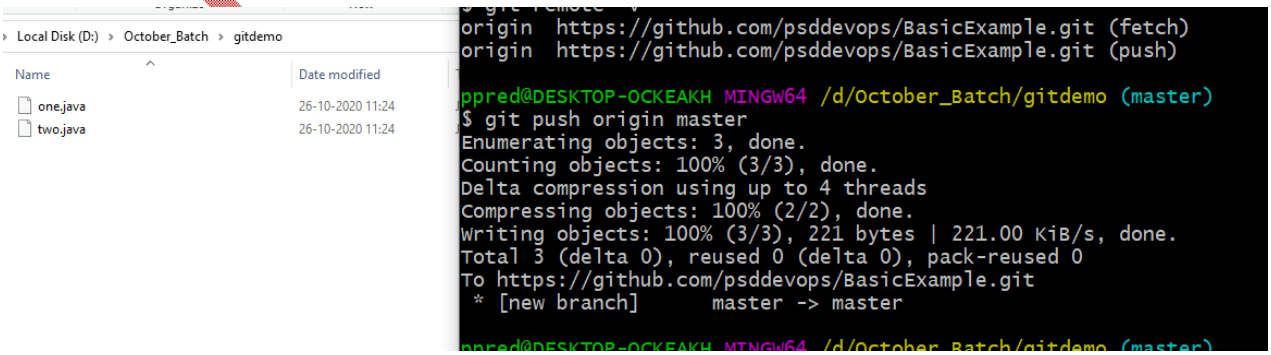
git remote add origin https://github.com/psddevops/BasicExample.git



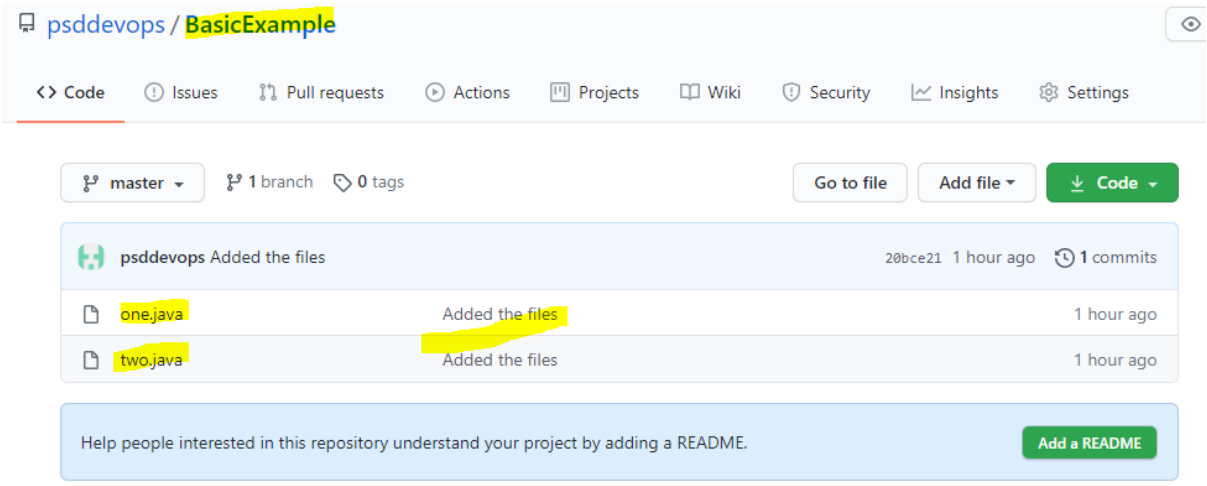
Push the changes in your local repository to GitHub remote repository.

git push origin master

Here push is the git command, origin is the remote name and master is the branch name.



Now our code is pushed to the remote repository.



git remote show origin :

It will give the information on a particular remote (here origin is the remote name)

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/psddevops/BasicExample.git
  Push  URL: https://github.com/psddevops/BasicExample.git
  HEAD branch: master
  Remote branch:
    master tracked
  Local ref configured for 'git push':
    master pushes to master (up to date)

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git status
On branch master
nothing to commit, working tree clean

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

git remote remove origin: It will remove the remote origins.

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git remote remove origin

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git remote -v

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Checking specific commit-id

git show <commit-id>

git show 21bdc50

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git show 21bdc50
commit 21bdc50f4af7a17550d5265f1d081c1fa5d9275b (HEAD -> master, origin/master)
Author: psddevops <ppreddy2711@gmail.com>
Date:   Mon Oct 26 12:27:41 2020 +0530

    Added four.java file

diff --git a/four.java b/four.java
new file mode 100644
index 0000000..e69de29

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Remove the files from working area/working directory

By using git clean we are going to remove the files in working directory/working area.

#git clean -n

Display the list of expected files

#git clean -f

Delete the files in working area

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ touch six.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        six.java

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

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git clean -n
Would remove six.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git clean -f
Removing six.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git status
On branch master
nothing to commit, working tree clean

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

git reset modes in GIT

Whenever we are modify and added the file into staging area the temporary-id will create repository that time head reference will pointing to the temporary-id.

There are *3-types* of modes in git reset.

- git reset --soft
- git reset –mixed
- git reset --hard

Note: Don't use reset if your commit is pushed to the remote, it causes problems.

git reset --soft

Move the head pointer from temporary-id to commit-id.

git reset --mixed

Move the head pointer and reset the staging area.

git reset --hard

Move the head pointer also reset the staging area and working area.

Ex: git reset --hard demo

ls

git log --oneline

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ ls
five.java  four.java  one.java  seven.java  six.java  three.java  two.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline
9317098 (HEAD -> master) Added seven.java
55894f7 Added six.java
583b2a0 Added five.java
21bdc50 (origin/master) Added four.java file
0d9003d Added three.java
```

Reverting the changes to four.java

git reset --hard 21bdc50

ls

git status

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git reset --hard 21bdc50
HEAD is now at 21bdc50 Added four.java file

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ ls
four.java  one.java  three.java  two.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Reverting the changes to seven.java

git reset --hard 9317098

git log –oneline

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git reset --hard 9317098
HEAD is now at 9317098 Added seven.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git status
On branch master
nothing to commit, working tree clean

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline
9317098 (HEAD -> master) Added seven.java
55894f7 Added six.java
583b2a0 Added five.java
21bdc50 (origin/master) Added four.java file
0d9003d Added three.java
20bce21 Added the files
```

Note: Whenever we are using reset hard make sure that we can copy and store hash value again we need to back easily.

git reset --soft demo

#ls

git log --oneline

git reset --soft 583b2a0

git status

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ ls
five.java  four.java  one.java  seven.java  six.java  three.java  two.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline
d027b7e (HEAD -> master) seven.java
55894f7 Added six.java
583b2a0 Added five.java
21bdc50 (origin/master) Added four.java file
0d9003d Added three.java
20bce21 Added the files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git reset --soft 583b2a0

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   seven.java
        new file:   six.java


ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

git reset --mixed demo

ls

#git log --oneline

git reset --mixed 583b2a0

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ ls
five.java  four.java  one.java  seven.java  six.java  three.java  two.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline
d027b7e (HEAD -> master) seven.java
55894f7 Added six.java
583b2a0 Added five.java
21bdc50 (origin/master) Added four.java file
0d9003d Added three.java
20bce21 Added the files
```

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git reset --mixed 583b2a0

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        seven.java
        six.java

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

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Reverting the Specific commit-id

git revert

If we want to remove specific commit we can use git revert. It doesn't remove the commit from the history; instead it reverts changes to the files and makes a new commit. This commit-id's will updated in the remote repository and other users also can visible this messages.

Note:

If commits are already pushed to remote we can't use reset instead we should go with revert.

#git log --oneline –all

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline --all
d027b7e (HEAD -> master) seven.java
55894f7 Added six.java
583b2a0 Added five.java
21bdc50 (origin/master) Added four.java file
0d9003d Added three.java
20bce21 Added the files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

#git revert 583b2a0

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git status
On branch master
nothing to commit, working tree clean

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git revert 583b2a0
Removing five.java
[master 4cb98f2] Revert "Added five.java"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 five.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git status
On branch master
nothing to commit, working tree clean

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ ls
four.java  one.java  seven.java  six.java  three.java  two.java
```

#git log

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log
commit 4cb98f2650136d40a70b3bac45964f8ffa078cbd (HEAD -> master)
Author: psddevops <ppreddy2711@gmail.com>
Date:   Tue Oct 27 10:23:30 2020 +0530

    Revert "Added five.java"

    This reverts commit 583b2a03de69295ac2cbcc1e7557f4313f39b979.

commit d027b7e60a76b143e6ce901d8d5d2b0c6f79d128
Author: psddevops <ppreddy2711@gmail.com>
Date:   Mon Oct 26 16:30:55 2020 +0530

    seven.java
```

Restore the changes

#git cherry-pick 583b2a03de69295ac2cbcc1e7557f4313f39b979

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git cherry-pick 583b2a03de69295ac2cbcc1e7557f4313f39b979
[master 3598bde] Added five.java
 Date: Mon Oct 26 16:05:44 2020 +0530
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 five.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ ls
five.java  four.java  one.java  seven.java  six.java  three.java  two.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

git cherry-pick

It's used for restoring the commit-id and also It picks a commit from different branch and applies it to current branch.

Syntax:

git cherry-pick <commit-id>

git checkout

If we want to traverse one commit-id to other commit-id we can go for git checkout.

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git checkout 55894f7
Note: switching to '55894f7'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 55894f7 Added six.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo ((55894f7...))
$
```

touch eight.java

git status

git add . && git commit -m "Added eight.java"

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo ((55894f7...))
$ touch eight.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo ((55894f7...))
$ git status
HEAD detached at 55894f7
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        eight.java

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

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo ((55894f7...))
$ git add . && git commit -m "Added eight.java"
[detached HEAD 8f53c1d] Added eight.java
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 eight.java
```

git log –oneline

ll

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo ((8f53c1d...))
$ git log --oneline
8f53c1d (HEAD) Added eight.java
55894f7 Added six.java
583b2a0 Added five.java
21bdc50 (origin/master) Added four.java file
0d9003d Added three.java
20bce21 Added the files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo ((8f53c1d...))
$ ll
total 0
-rw-r--r-- 1 ppred 197609 0 Oct 27 10:47 eight.java
-rw-r--r-- 1 ppred 197609 0 Oct 27 10:29 five.java
-rw-r--r-- 1 ppred 197609 0 Oct 26 12:27 four.java
-rw-r--r-- 1 ppred 197609 0 Oct 26 11:24 one.java
-rw-r--r-- 1 ppred 197609 0 Oct 26 16:53 six.java
-rw-r--r-- 1 ppred 197609 0 Oct 26 12:22 three.java
-rw-r--r-- 1 ppred 197609 0 Oct 26 11:24 two.java
```

git checkout master

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo ((8f53c1d...))
$ git checkout master
Warning: you are leaving 1 commit behind, not connected to
any of your branches:

  8f53c1d Added eight.java

If you want to keep it by creating a new branch, this may be a good time
to do so with:

 git branch <new-branch-name> 8f53c1d

Switched to branch 'master'
```

git log --oneline

ls

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline
3598bde (HEAD -> master) Added five.java
4cb98f2 Revert "Added five.java"
d027b7e seven.java
55894f7 Added six.java
583b2a0 Added five.java
21bdc50 (origin/master) Added four.java file
0d9003d Added three.java
20bce21 Added the files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ ls
five.java  four.java  one.java  seven.java  six.java  three.java  two.java
```

git cherry-pick 8f53c1d

git log –oneline

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git cherry-pick 8f53c1d
[master 69ca6b7] Added eight.java
 Date: Tue Oct 27 10:48:13 2020 +0530
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 eight.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline
69ca6b7 (HEAD -> master) Added eight.java
3598bde Added five.java
4cb98f2 Revert "Added five.java"
d027b7e seven.java
55894f7 Added six.java
583b2a0 Added five.java
21bdc50 (origin/master) Added four.java file
0d9003d Added three.java
20bce21 Added the files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Branches

Branch is used to work on a specific task (enhancement, bug, new feature) Branch provides isolation, i.e. other work will not impact my work.

Branch

- Branches are used to create another line of development.
- By default, Git has a master branch, which is same as trunk in Subversion (SVN). Usually, a branch is created to work on a new feature.
- Once the feature is completed, it is merged back with the master branch and we delete the branch.

Master Branch:

- Every git repository comes with a default branch which is called as master branch.
  - No one should directly work on master.
  - Master must contain only well tested code.
  - In real world most of the guys will not have permissions to push to master.
- If there is a new task to work on, then we should create a branch and work on it.

Tags

- Tags similar to branches, but the difference is that tags are immutable.
- It means, tag is a branch, which nobody intends to modify. Once a tag is created for a particular commit, even if you create a new commit, it will not be updated.
- Usually, developers create tags for product releases.

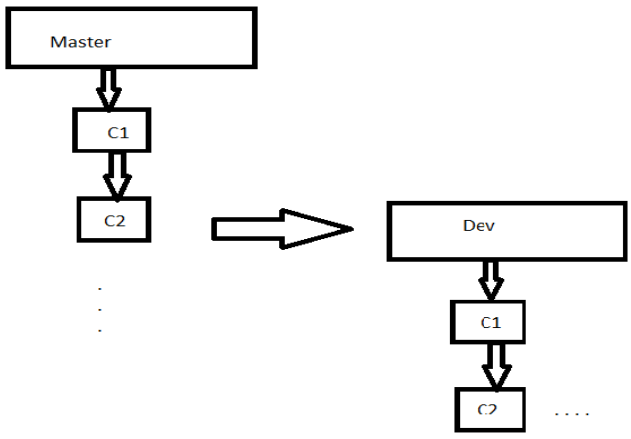
Creating Branch

By using below command to create a new branch from current branch, whenever we are creating branch all commit-id data up to date its present.

Syntax

git branch <branch-name>

git branch Dev



Creating dev Brach from master

git branch dev

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch
* master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch dev

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch
  dev
* master
```

Switching the branch from master to dev

 # git checkout dev

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git checkout dev
Switched to branch 'dev'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$
```

Verifying the logs in master and dev branches

 # git log --oneline

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$ git log --oneline
21bdc50 (HEAD -> dev, origin/master, master) Added four.java file
0d9003d Added three.java
20bce21 Added the files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$ git checkout master
Switched to branch 'master'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline
21bdc50 (HEAD -> master, origin/master, dev) Added four.java file
0d9003d Added three.java
20bce21 Added the files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Note:

(HEAD -> master, origin/master, dev) → master && dev branches having same commit-id, origin/master(remote repo) also having commit-id.

Creating a file in dev branch and merge to the master branch

git checkout dev

touch five.java

git add . && git commit -m "Added five.java in dev branch"

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$ touch five.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$ git add . && git commit -m "Added five.java in dev branch"
[dev 3ed9064] Added five.java in dev branch
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 five.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$ git log --oneline --all
3ed9064 (HEAD -> dev) Added five.java in dev branch
21bdc50 (origin/master, master) Added four.java file
0d9003d Added three.java
20bce21 Added the files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$
```

In above diagram

- (HEAD -> dev) is pointing latest commit-id i.e. "3ed9064"
- (origin/master, master) is pointing previous commit-id i.e. "21bdc50"

Merging to master branch

Switch to master branch and verify the log

#git checkout master

#git log --oneline --all

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$ git checkout master
Switched to branch 'master'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline --all
3ed9064 (dev) Added five.java in dev branch
21bdc50 (HEAD -> master, origin/master) Added four.java file
0d9003d Added three.java
20bce21 Added the files
```

Use git cherry-pick command merge to the master branch

#git cherry-pick 3ed9064

#git log --oneline --all

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git cherry-pick 3ed9064
[master 11a0f65] Added five.java in dev branch
 Date: Tue Oct 27 13:04:07 2020 +0530
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 five.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline --all
11a0f65 (HEAD -> master) Added five.java in dev branch
3ed9064 (dev) Added five.java in dev branch
21bdc50 (origin/master) Added four.java file
0d9003d Added three.java
20bce21 Added the files
```

In above diagram master and dev branches pointing to same file.

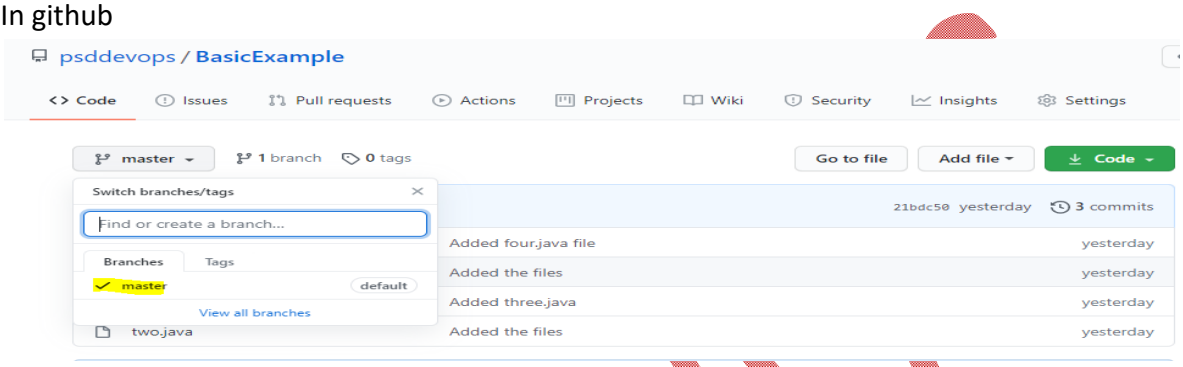
Origin/master pointing to different commit-id

Pushing branches to remote repository (github)

Find out the branches in remote repo

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch
  dev
* master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch -r
  origin/master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

In github



Pushing dev branch

git checkout dev

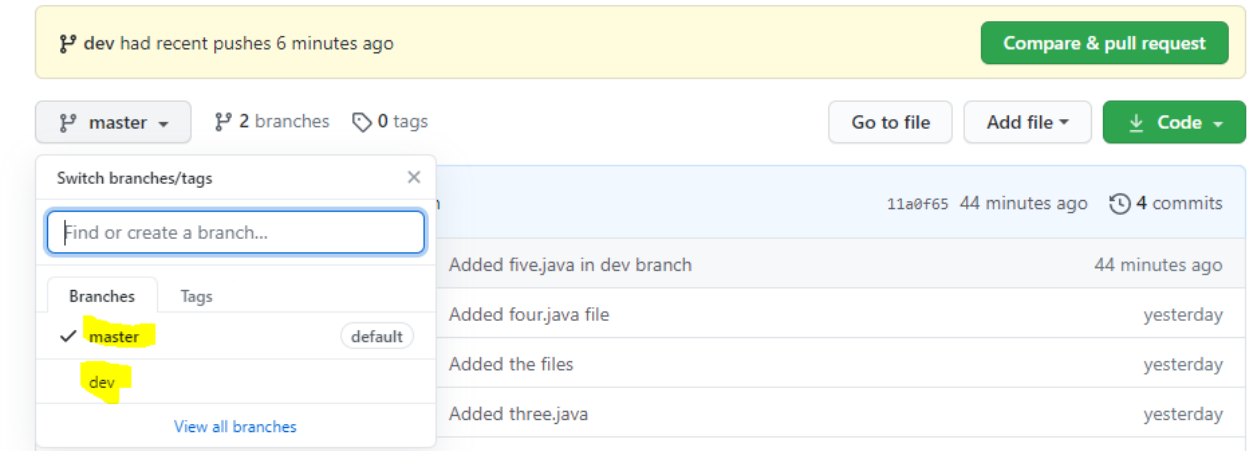
git push origin dev

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch -r
  origin/master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git checkout dev
Switched to branch 'dev'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$ git push origin dev
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 250 bytes | 125.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:      https://github.com/psddevops/BasicExample/pull/new/dev
remote:
To https://github.com/psddevops/BasicExample.git
 * [new branch]      dev -> dev

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$ git branch -r
  origin/dev
  origin/master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$
```

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$ git log --oneline --all
11a0f65 (master) Added five.java in dev branch
3ed9064 (HEAD -> dev, origin/dev) Added five.java in dev branch
21bdc50 (origin/master) Added four.java file
0d9003d Added three.java
20bce21 Added the files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$
```

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (dev)
$ git checkout master
Switched to branch 'master'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git push -u --all
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 255 bytes | 255.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/psddevops/BasicExample.git
   21bdc50..11a0f65  master -> master
Branch 'dev' set up to track remote branch 'dev' from 'origin'.
Branch 'master' set up to track remote branch 'master' from 'origin'.

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log --oneline --all
11a0f65 (HEAD -> master, origin/master) Added five.java in dev branch
3ed9064 (origin/dev, dev) Added five.java in dev branch
21bdc50 Added four.java file
0d9003d Added three.java
20bce21 Added the files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
```

By using "git push -u –all" we can push all branches to the remote repo



Using merge/rebase command to merge the branches

Merge demo

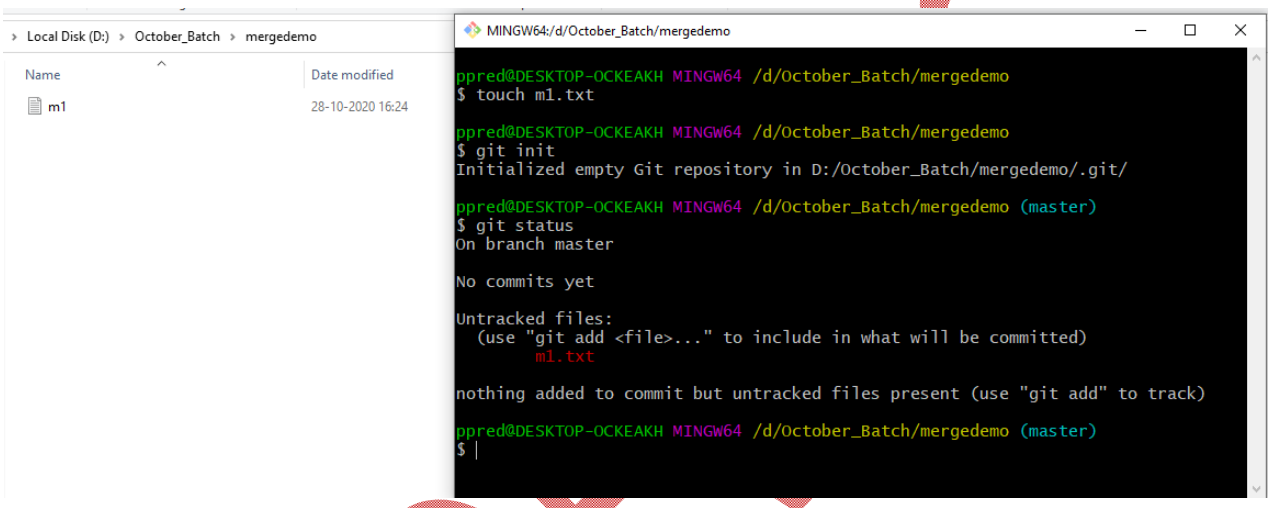
Git merge is used integrate changes from one branch into another branch.

Create a project mergedemo and create master and dev branches

git init

touch m1.txt

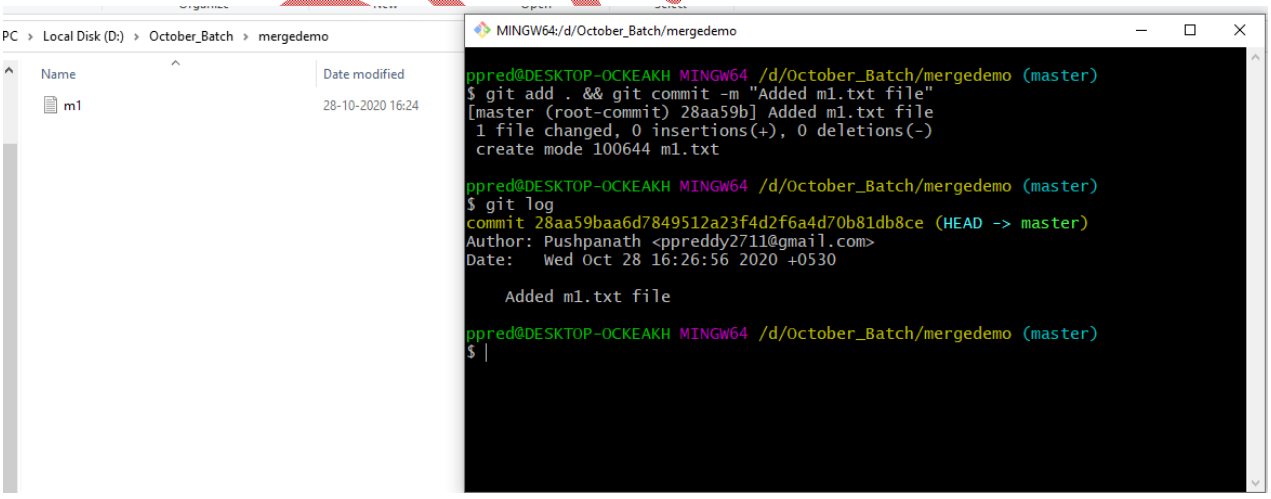
git status



Added to staging and commit

git add . && git commit -m "Added m1.txt file"

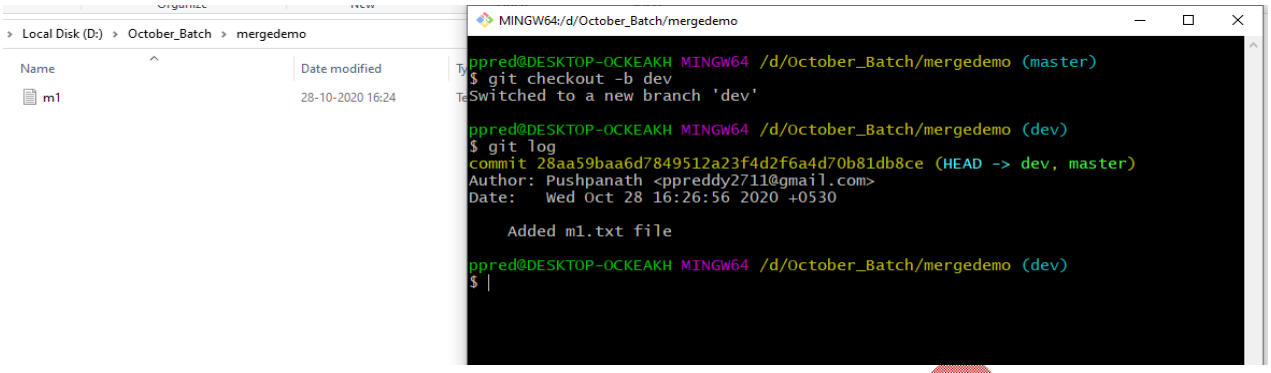
git log



Create and checkout to dev branch

git checkout -b dev

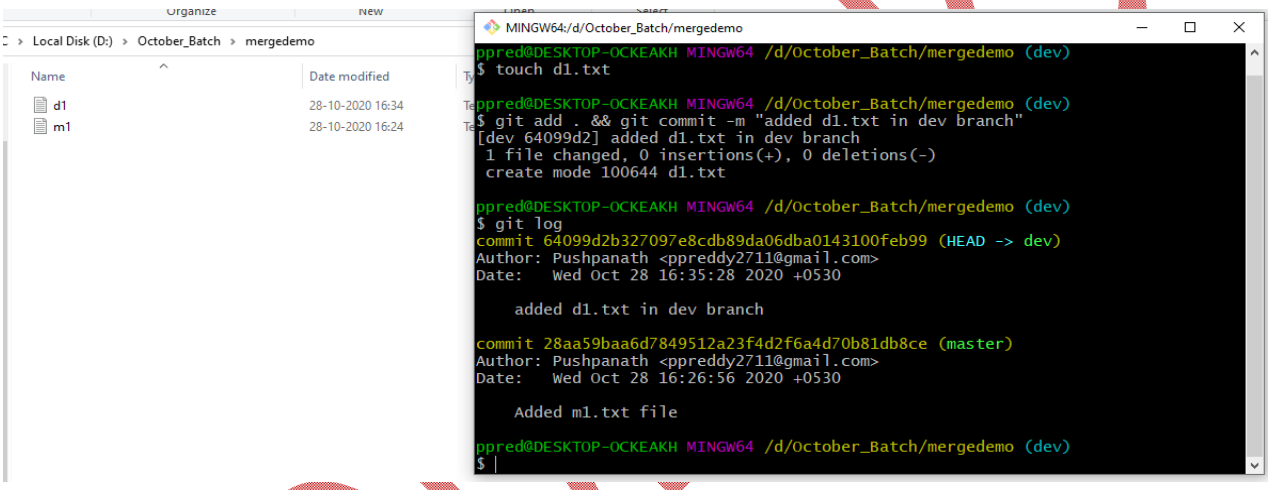
git log



Create d1.txt file dev branch

#touch d1.txt

#git add . && git commit -m "added d1.txt in dev branch"



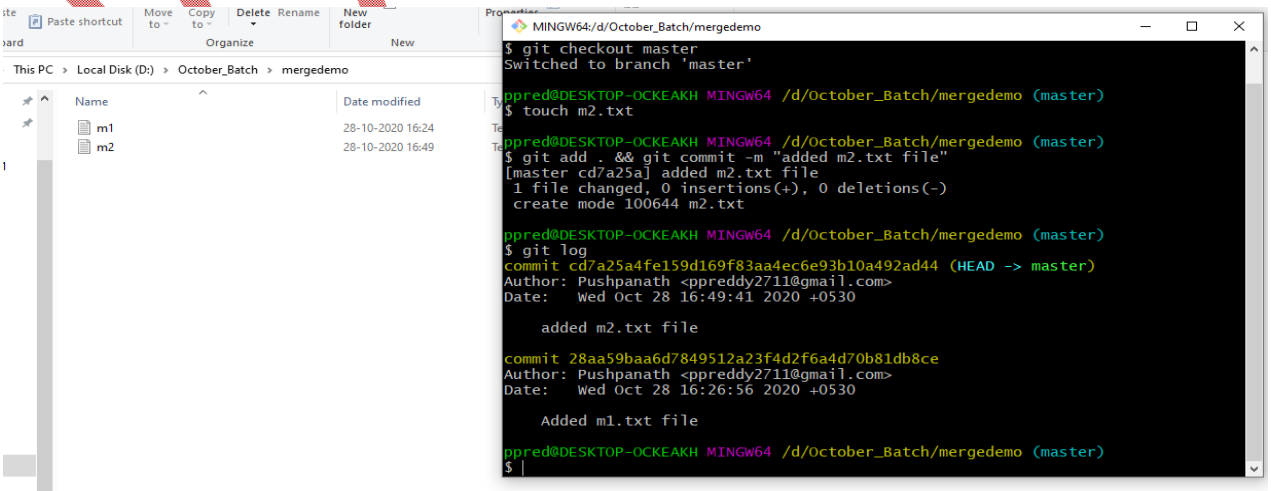
Checkout to master and create m2.txt file

git checkout master

touch m2.txt

git add . && git commit -m "added m2.txt file"

git log

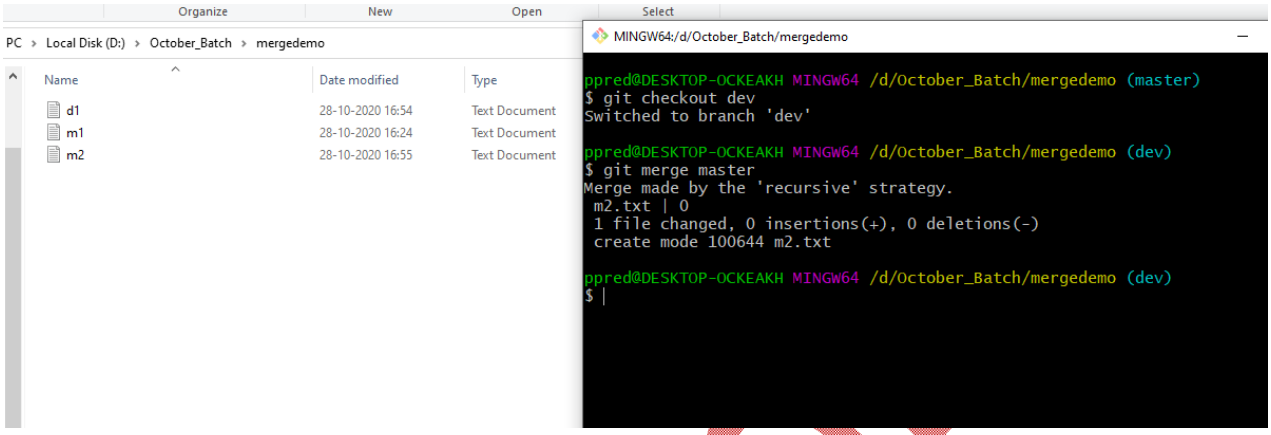


*Prepared by **PPNREDDY** .....*

Checkout to dev and merge to master

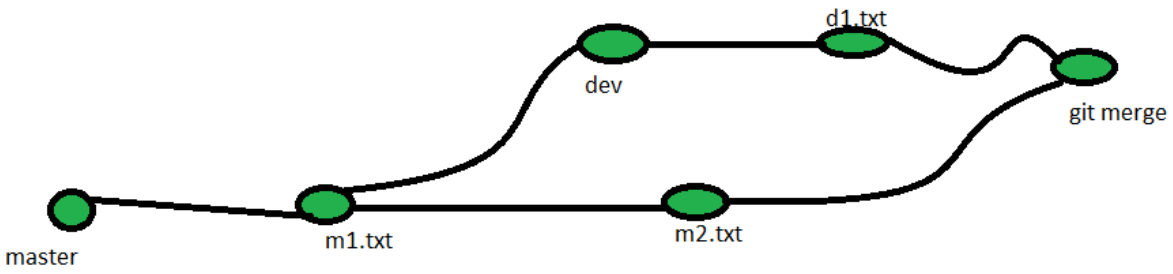
git checkout dev

git merge master



#git log

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/mergedemo (dev)
$ git log
commit 1a0b54f5cfe50a6950efb49c3bd1b659f78948db (HEAD -> dev)
Merge: 64099d2 cd7a25a
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Oct 28 16:55:10 2020 +0530

    Merge branch 'master' into dev

commit cd7a25a4fe159d169f83aa4ec6e93b10a492ad44 (master)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Oct 28 16:49:41 2020 +0530

    added m2.txt file

commit 64099d2b327097e8cdb89da06dba0143100feb99
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Oct 28 16:35:28 2020 +0530

    added d1.txt in dev branch

commit 28aa59baa6d7849512a23f4d2f6a4d70b81db8ce
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Oct 28 16:26:56 2020 +0530

    Added m1.txt file

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/mergedemo (dev)
$
```

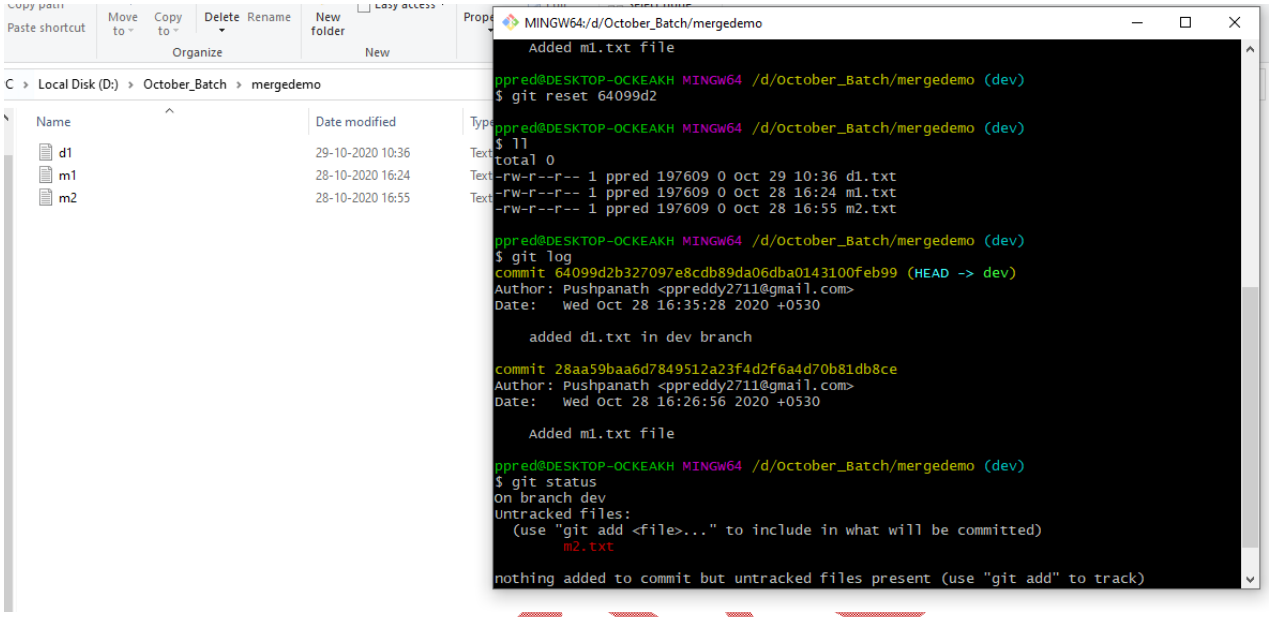


*Prepared by **PPNREDDY** .....*

Reverting the merge

git reset 64099d2

git log



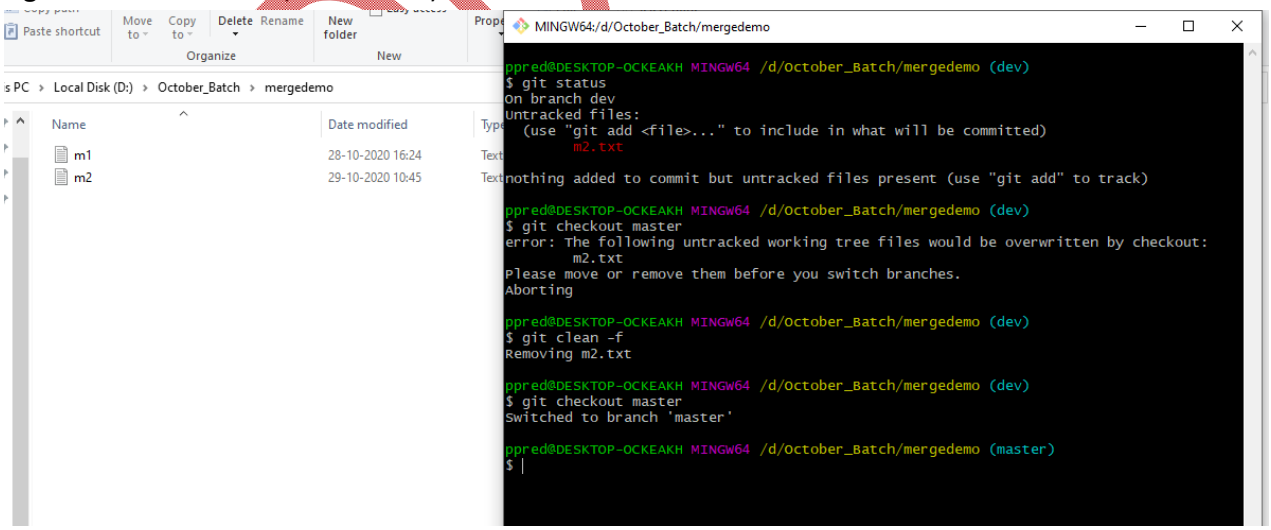
If we are trying to checkout master branch will get error need to delete the working area files.

git checkout master (It will get error)

git clean –n

git clean –f

git checkout master (It will work)



Verify the logs in master and dev branch.

git log

git log

*Prepared by **PPNREDDY** .....*

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/mergedemo (master)
$ git log
commit cd7a25a4fe159d169f83aa4ec6e93b10a492ad44 (HEAD -> master)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Oct 28 16:49:41 2020 +0530

    added m2.txt file

commit 28aa59baa6d7849512a23f4d2f6a4d70b81db8ce
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Oct 28 16:26:56 2020 +0530

    Added m1.txt file

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/mergedemo (master)
$ git checkout dev
Switched to branch 'dev'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/mergedemo (dev)
$ git log
commit 64099d2b327097e8cdb89da06dba0143100feb99 (HEAD -> dev)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Oct 28 16:35:28 2020 +0530

    added d1.txt in dev branch

commit 28aa59baa6d7849512a23f4d2f6a4d70b81db8ce
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Oct 28 16:26:56 2020 +0530

    Added m1.txt file

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/mergedemo (dev)
$
```

Rebase demo

Git rebase is used integrate changes from one branch into another branch.

Create a project rebasedemo and create m1.txt file in master branch.

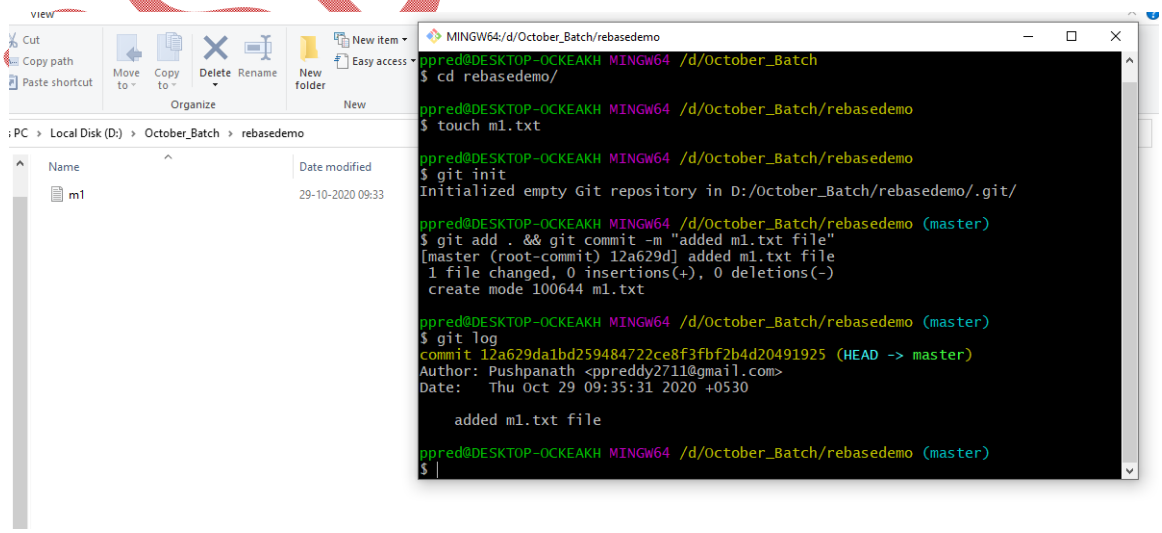
mkdir rebasedemo

touch m1.txt

git init

git add . && git commit -m "added m1.txt file"

git log



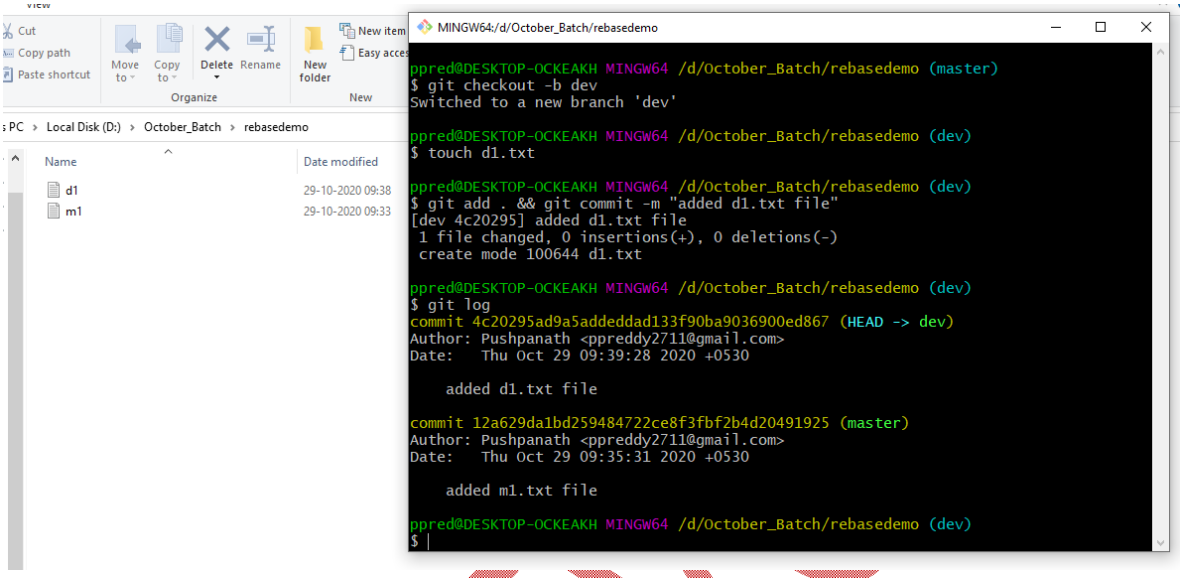
Create dev branch and add d1.txt file

git checkout -b dev

touch d1.txt

git add . && git commit -m "added d1.txt file"

git log



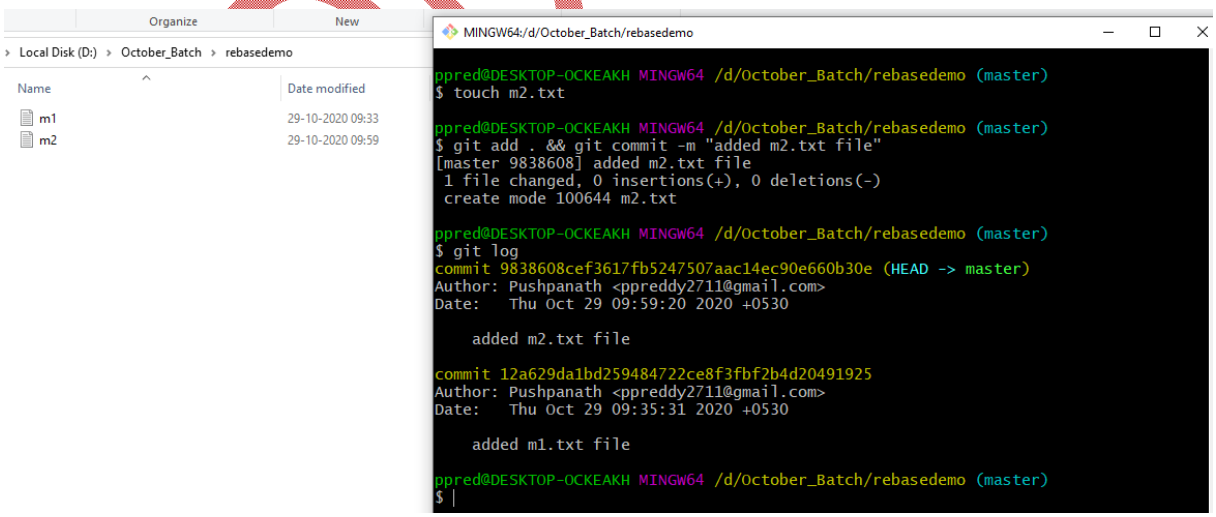
Create m2.txt in master and merge to the dev branch using rebase process

git checkout master

touch m2.txt

git add . && git commit -m "added m2.txt file"

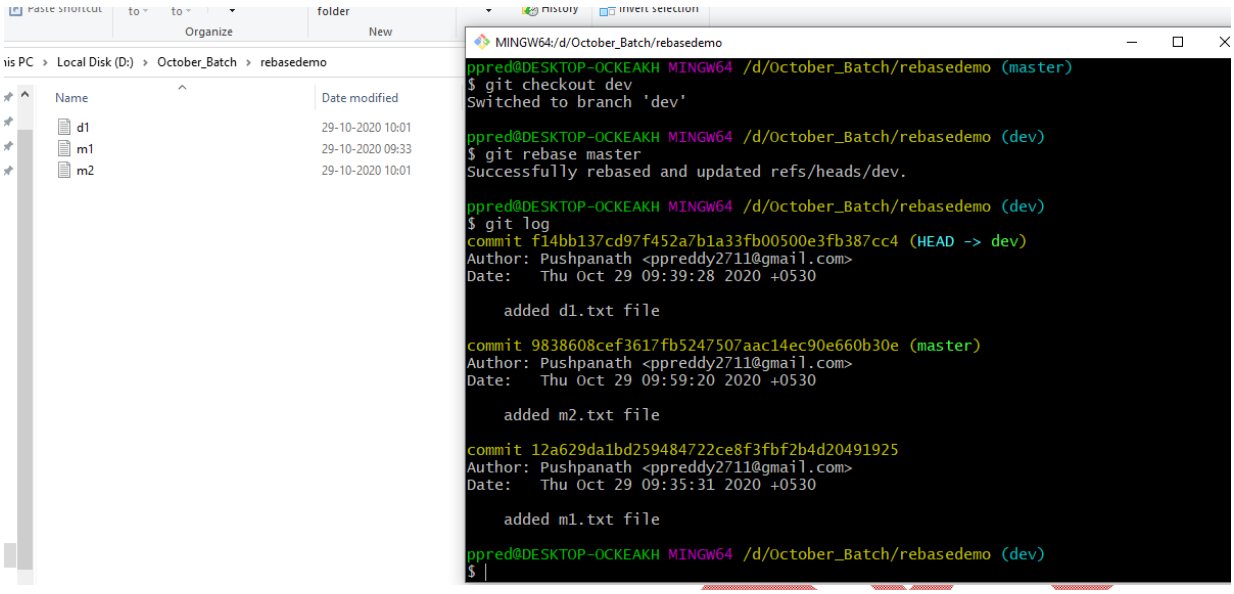
git log

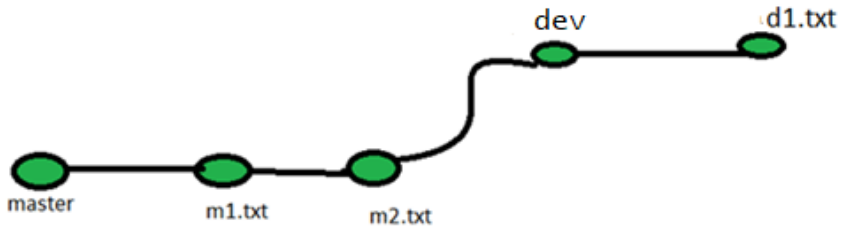


git checkout dev

git rebase master

git log





git merge Vs rebase

Git *rebase* and *merge* both integrate changes from one branch into another

git merge

- Is a non-destructive operation.
- Existing branches are not changed in any way.
- Creates a new merge commit in the feature branch.

git rebase

- Moves the entire feature branch to begin on the tip of the master branch.
- Re-writes the project history.
- We get much cleaner and linear project history.

Note: Use merge if we have any issue we can reset easily.

Git branching strategy

Branching strategies coordinate work to allow for easier integration of changes and releases.The point of a branching strategy is to efficiently manage code changes. This workflow will impact both developer and deployment workflows. It's mainly divided into 2-categories.

1. Main branches
2. Supporting/Supplementary branches

Main branches

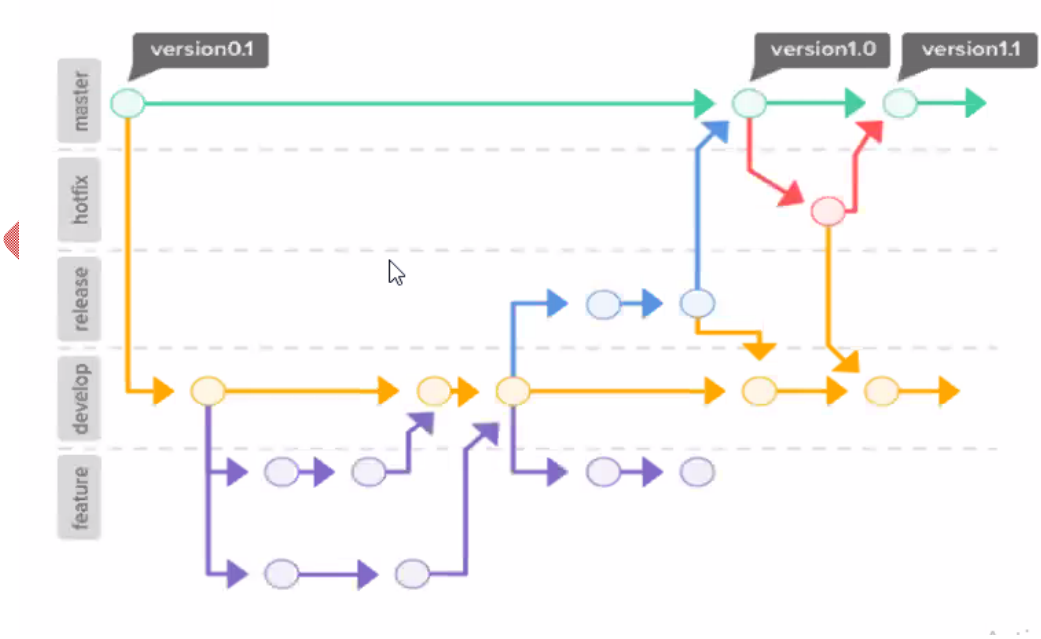
The development model is greatly inspired by existing models out there. The central repo holds two main branches with an infinite lifetime.

- master branch
- develop branch

Supporting/Supplementary branches

Supporting branches to aid parallel development between team members, ease tracking of features, prepare for production releases and to assist in quickly fixing live production problems. Unlike the main branches, these branches always have a limited life time, since they will be removed eventually.

- feature branch
- release branch
- bugfix branch
- hotfix branch



Stable Master Branch Tag Model

Master branch:

Master must contain well tested code, application release happens from this branch by creating a release 'tag'. All development code is merged into master in sometime. In real world no one directly work on master.

Develop branch:

This branch belongs to a specific team, code integration of this team members are done on this branch. Develop branch is created from master. This branch contains pre-production code. When the features are finished then they are merged into *develop* branch.

Feature branch:

Feature branch belongs to a specific developer, where his feature in implemented, after completion of a feature changes are merged into his develop branch. Feature branch is created from develop branch.

Release/UAT branch:

This branch is to integrate changes done by multiple teams under their develop branch. Release branches support preparation of a new production release. They allow many minor bug to be fixed and preparation of meta-data for a release. May branch off from *develop* and must merge into *master* and *develop*.

Bugfix branch:

This branch is used for fixing UAT defects.Bugfixes branch is created from *release*. Bugfix branches are necessary to act immediately upon an undesired status of *release*. May branch off from *release* and must merge into *release* branch.

Hotfix branch:

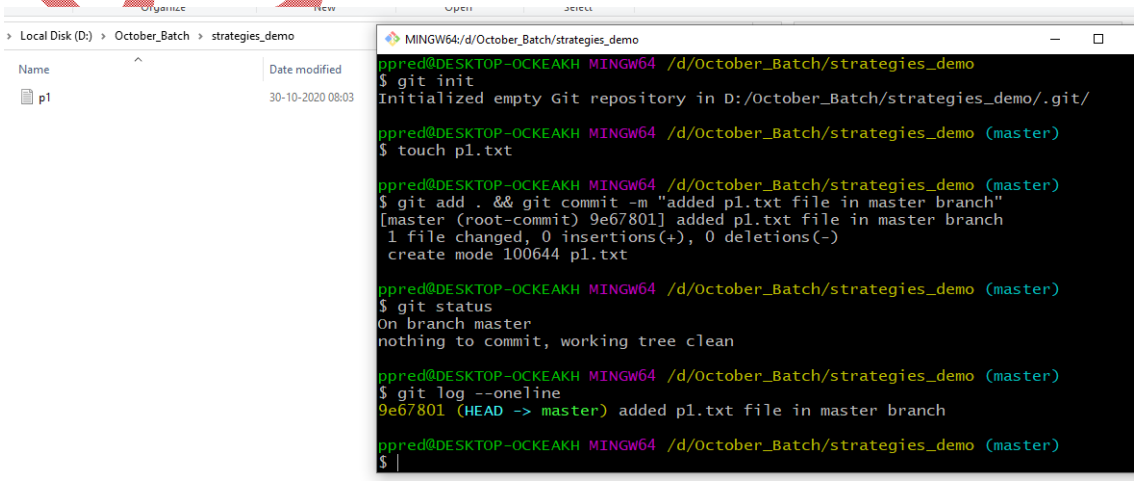
This branch is used for fixing production defects. Hotfixes branch is created from *master*. Hotfix branches are necessary to act immediately upon an undesired status of *master*. May branch off from *master* and must merge into *master* and *develop.*

Branching Strategies Demo

1. Create a file p1.txt in master branch which is currently running on PROD.

   # touch p1.txt

   # git add . && git commit -m "added p1.txt file in master branch"

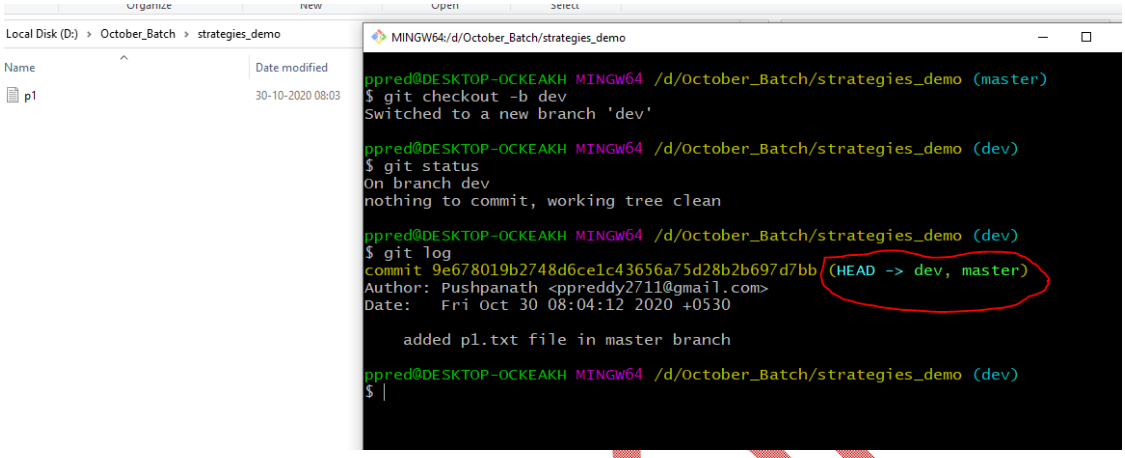


2. Create dev branch from master branch .i.e. p1.txt file available in dev branch also.

   # git checkout -b dev

   # git status

   # git log



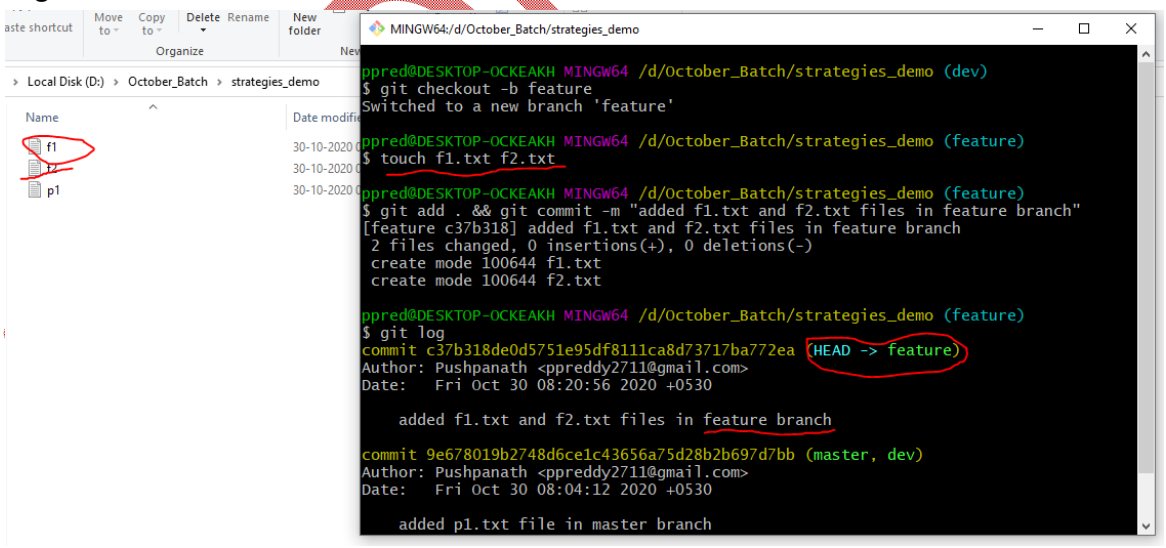
3. Create feature branch from dev and add f1.txt and f2.txt files in feature, merge and delete the feature branch to dev branch.

Creating files in feature branch

git checkout -b feature

touch f1.txt f2.txt

git log



Merging feature branch to dev branch and delete the feature branch

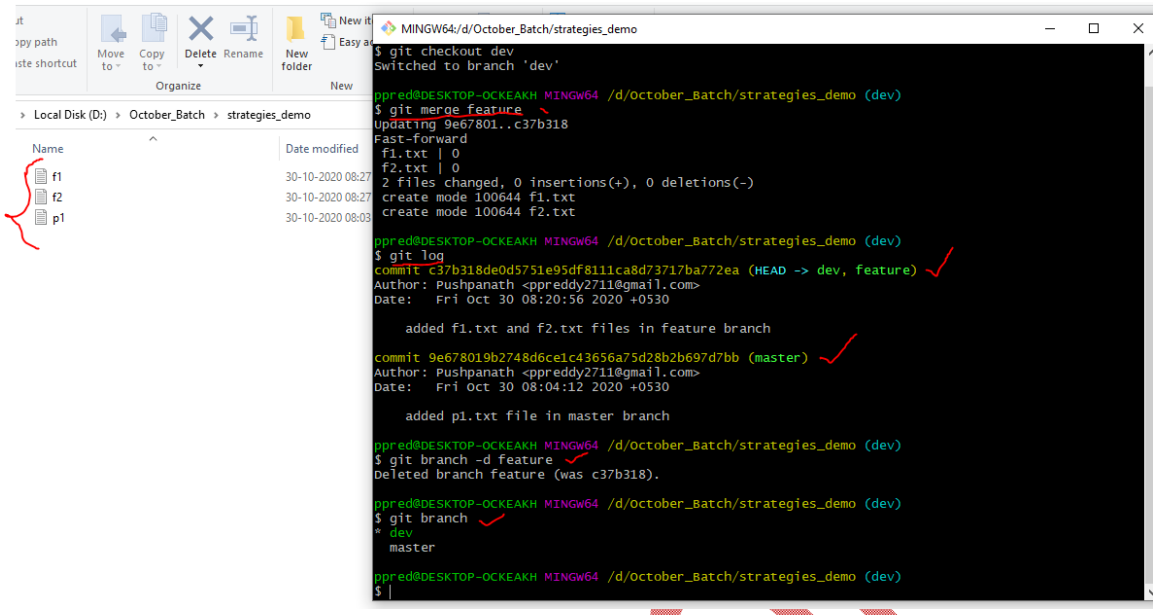
git checkout dev

git merge feature

git log

git branch -d feature (Deleting the feature branch)

git branch (List out the all branches)

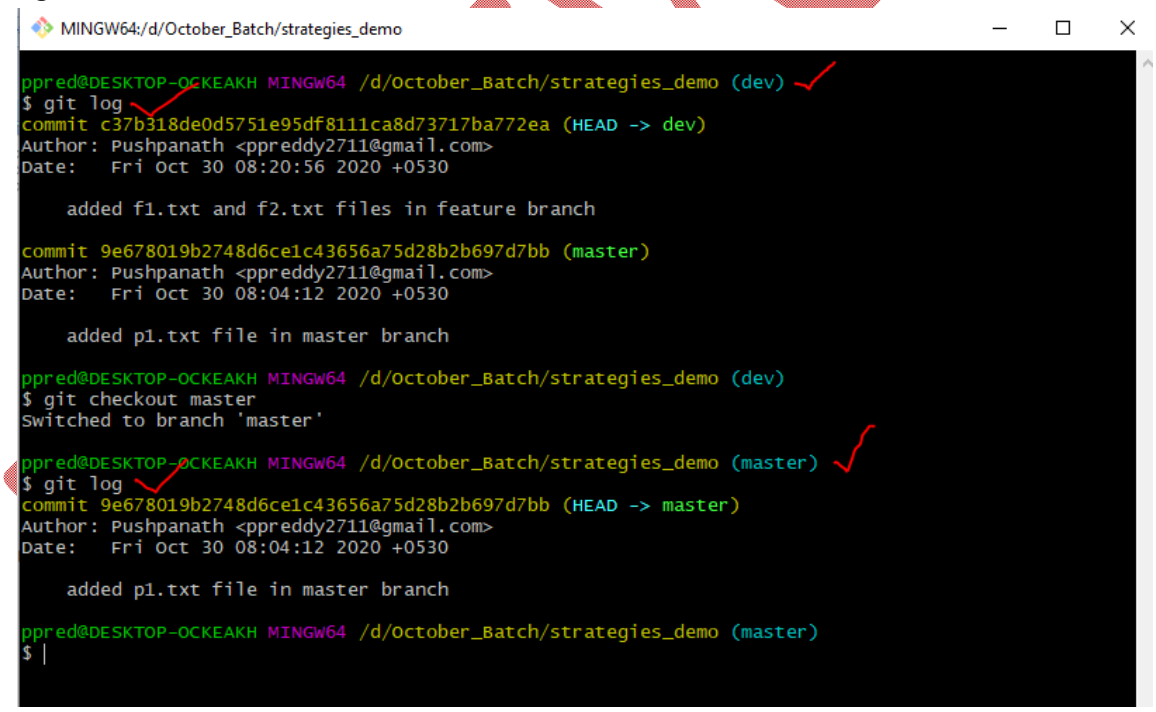


Verify the logs in dev and master branches

git log

git checkout master

git log

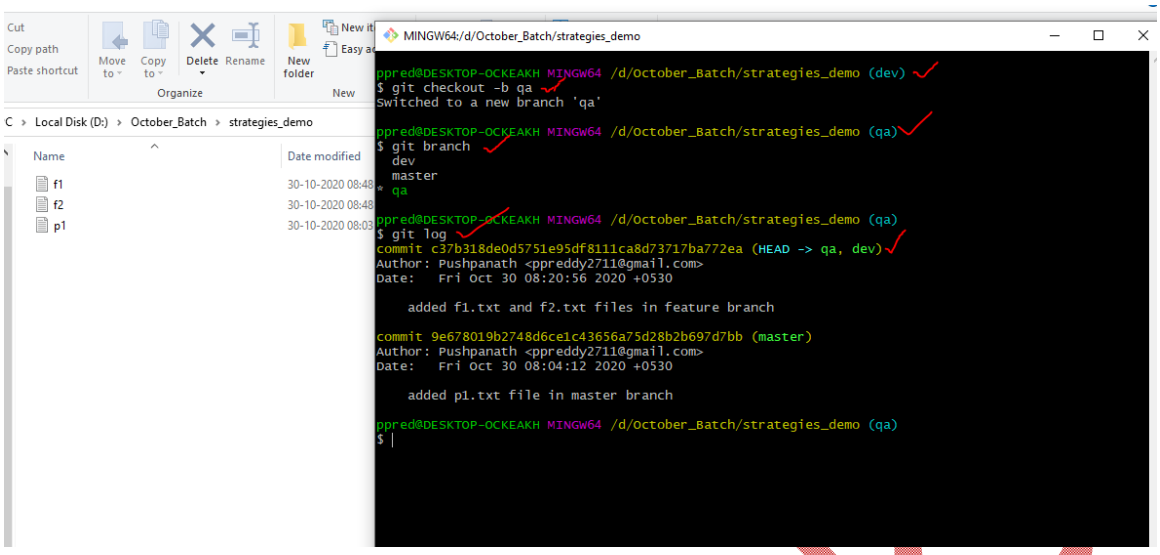


4. Create QA/release branch from dev branch for testing purpose.

git checkout dev

git checkout -b qa

git log



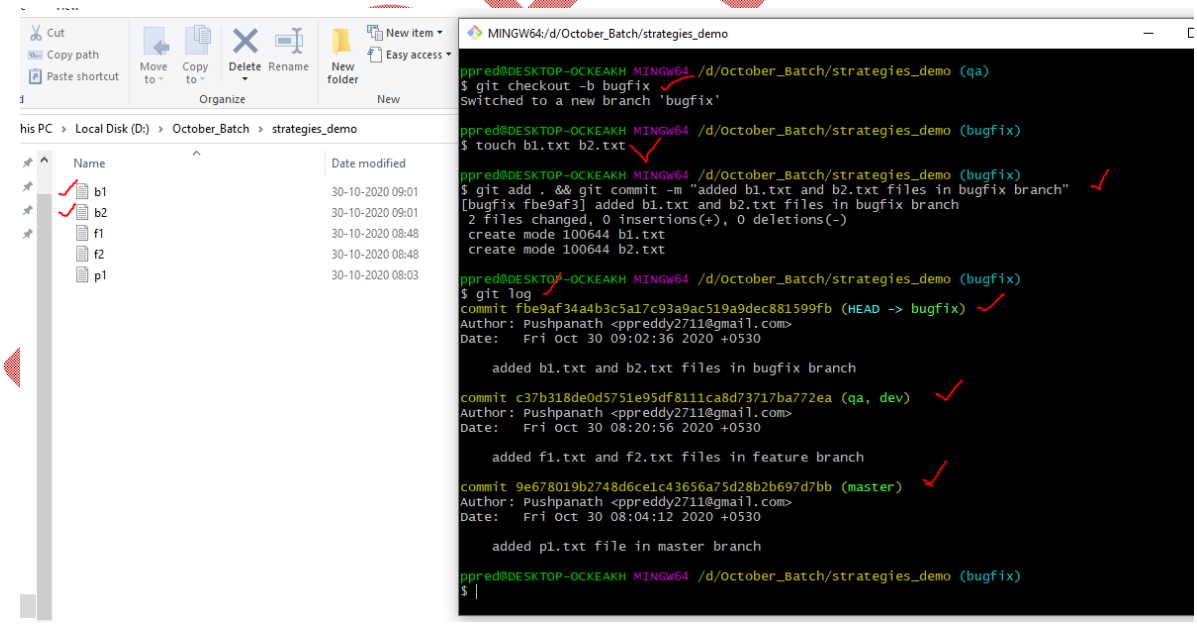
5. If we found any issues in qa testing we need to create bugfix branch from qa branch. Once issues will complete merge to the qa branch and delete the bugfix branch.

   # git checkout -b bugfix

   # touch b1.txt b2.txt

   # git add . && git commit -m "added b1.txt and b2.txt files in bugfix branch"

   # git log

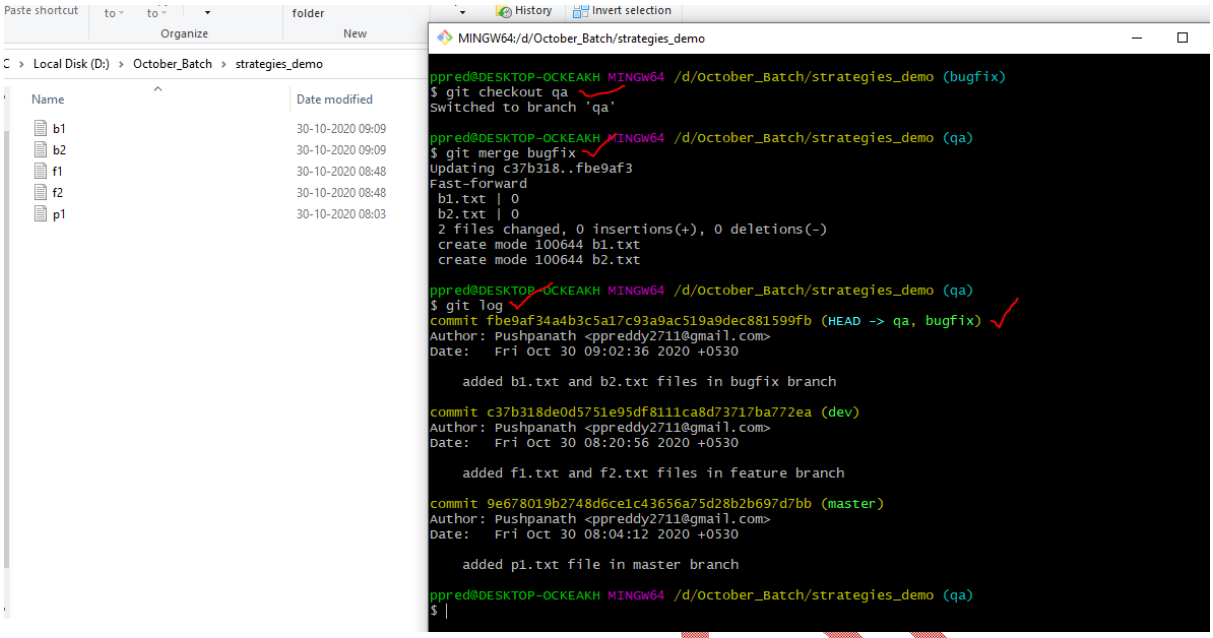


Merging bugfix branch to qa branch and delete the bugfix branch

git checkout qa

git merge bugfix

git log



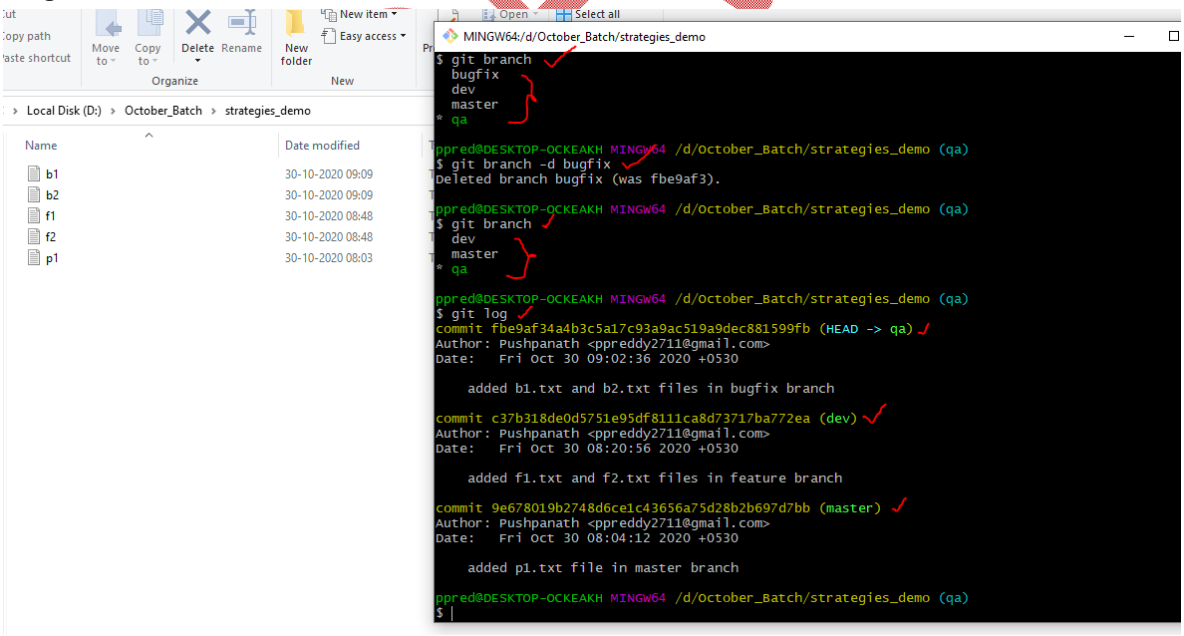
Deleting the bugfix branch

git branch

git branch -d bugfix

git branch

git log



6. Merge the qa branch to master and dev branches and delete the qa branch.

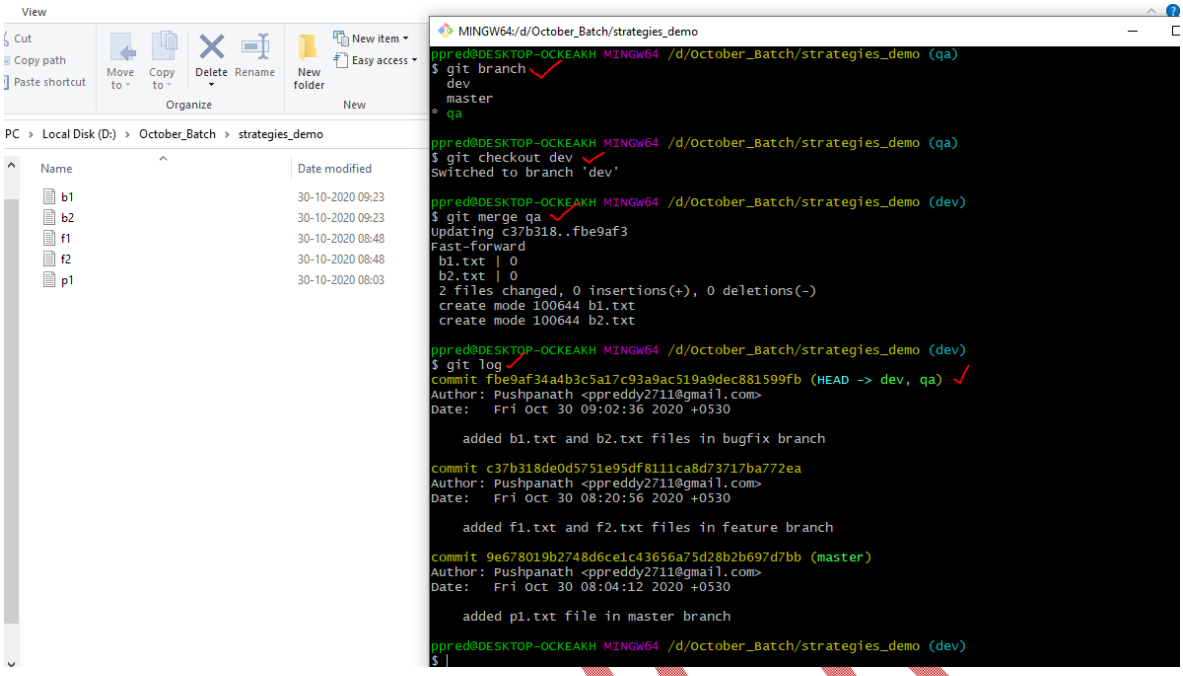
   Merging **qa** branch to the **dev** branch

   # git branch

   # git checkout dev

   # git merge qa

   # git log

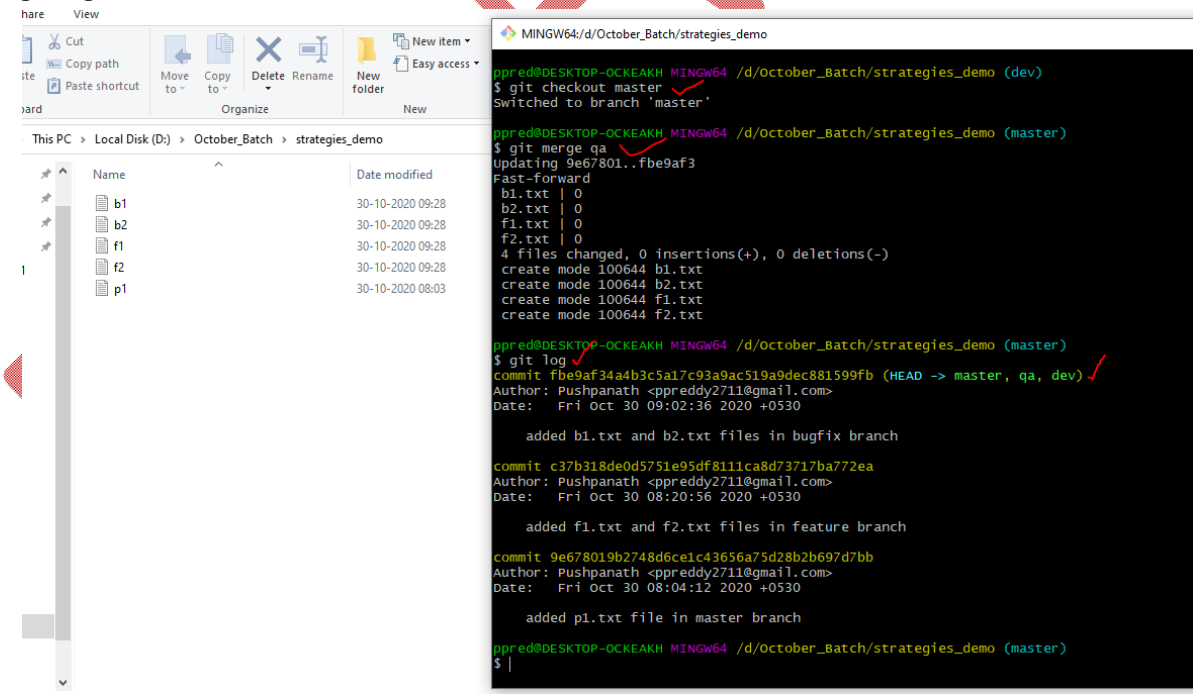


Merging **qa** branch to the **master** branch

git checkout master

git merge qa

git log



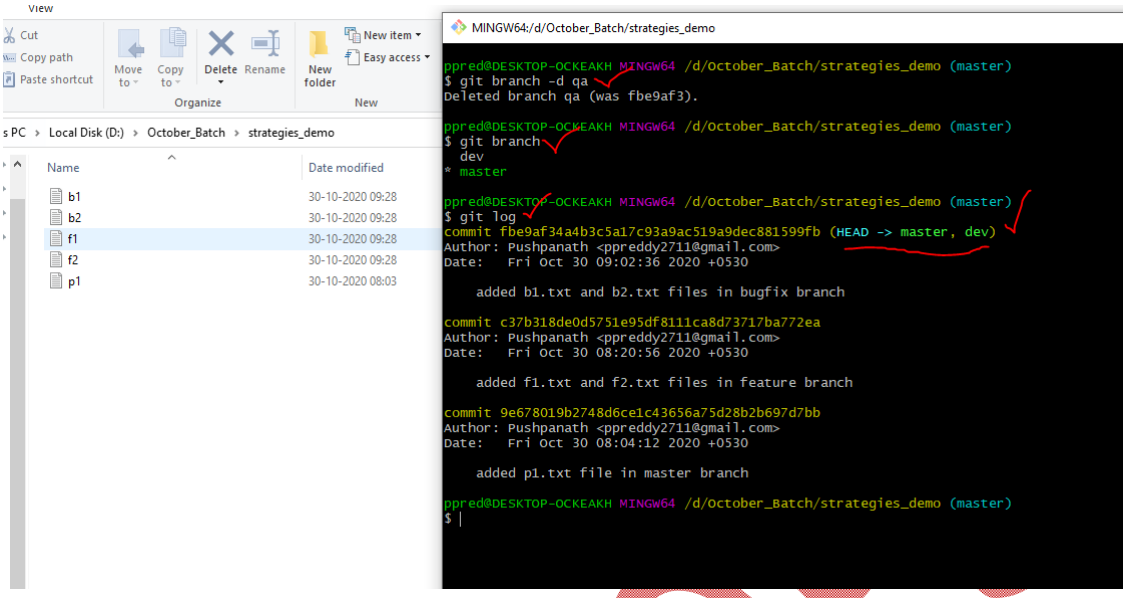
Deleting the qa branch and verify the master and dev branch

git branch -d qa

git branch

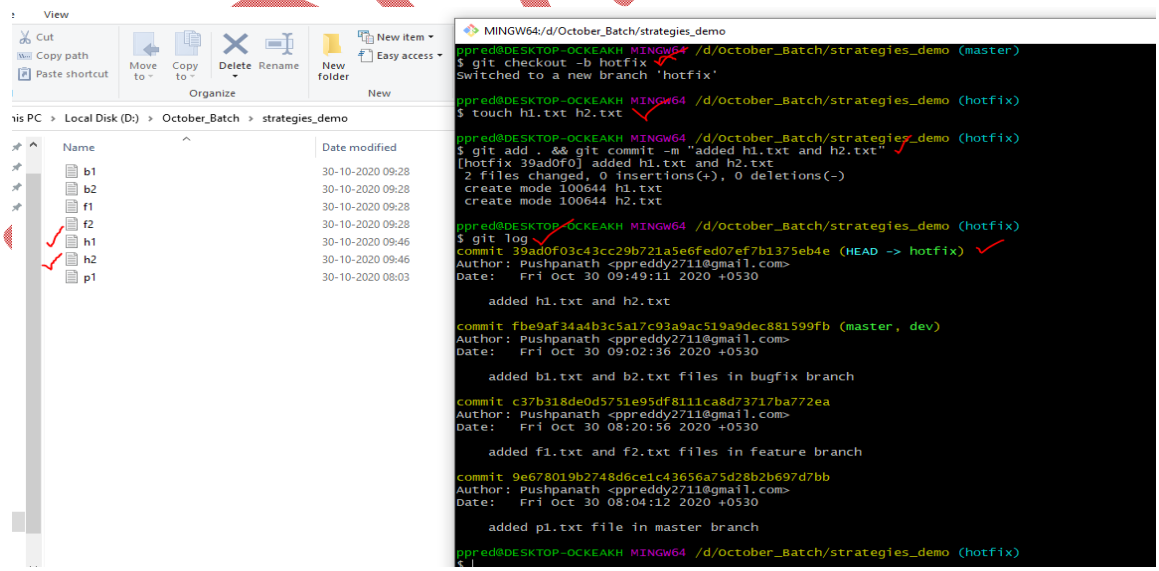
git log

*Prepared by **PPNREDDY** .....*



Now the code same in master and dev branch

7. If we found any issues in master (PROD) branch. We can create the hotfix branch from master and merge to the dev and master branches. Finally we can delete the hotfix branch.
   # git checkout -b hotfix
   # touch h1.txt h2.txt
   # git add . && git commit -m "added h1.txt and h2.txt"
   # git log



Merge and delete the hotfix branch to dev and master branches
git checkout master
git merge hotfix
git log

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/strategies_demo (hotfix)
$ git branch
  dev
* hotfix
  master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/strategies_demo (hotfix)
$ git checkout master
Switched to branch 'master'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/strategies_demo (master)
$ git merge hotfix
Updating fbe9af3..39ad0f0
Fast-forward
 h1.txt | 0
 h2.txt | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 h1.txt
 create mode 100644 h2.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/strategies_demo (master)
$ git log
commit 39ad0f03c43cc29b721a5e6fed07ef7b1375eb4e (HEAD -> master, hotfix)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Fri Oct 30 09:49:11 2020 +0530

    added h1.txt and h2.txt

commit fbe9af34a4b3c5a17c93a9ac519a9dec881599fb (dev)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Fri Oct 30 09:02:36 2020 +0530

    added b1.txt and b2.txt files in bugfix branch

commit c37b318de0d5751e95df8111ca8d73717ba772ea
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Fri Oct 30 08:20:56 2020 +0530

    added f1.txt and f2.txt files in feature branch

commit 9e678019b2748d6ce1c43656a75d28b2b697d7bb
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Fri Oct 30 08:04:12 2020 +0530

    added p1.txt file in master branch
```

git checkout dev

git merge hotfix

git branch -d hotfix

git log

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/strategies_demo (master)
$ git checkout dev
Switched to branch 'dev'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/strategies_demo (dev)
$ git merge hotfix
Updating fbe9af3..39ad0f0
Fast-forward
 h1.txt | 0
 h2.txt | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 h1.txt
 create mode 100644 h2.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/strategies_demo (dev)
$ git log -1
commit 39ad0f03c43cc29b721a5e6fed07ef7b1375eb4e (HEAD -> dev, master, hotfix)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Fri Oct 30 09:49:11 2020 +0530

    added h1.txt and h2.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/strategies_demo (dev)
$ git branch -d hotfix
Deleted branch hotfix (was 39ad0f0).

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/strategies_demo (dev)
$ git branch
* dev
  master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/strategies_demo (dev)
$ git log -1
commit 39ad0f03c43cc29b721a5e6fed07ef7b1375eb4e (HEAD -> dev, master)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Fri Oct 30 09:49:11 2020 +0530

    added h1.txt and h2.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/strategies_demo (dev)
$
```

Merge conflicts

When two branches are trying to merge, and both are edited at the same time and in the same file, Git won't be able to identify which version is to take for changes. Such a situation is called merge conflict. If such a situation occurs, it stops just before the merge commit so that you can resolve the conflicts manually.

1. Resolving merge conflicts manually
2. Resolving merge conflicts using merge tools.

Resolving merge conflicts manually

We are trying to merge one branch to another branch with same file the conflicts will come.By update the file with require content we will resolve the issue.

1. Create state.txt file in master branch

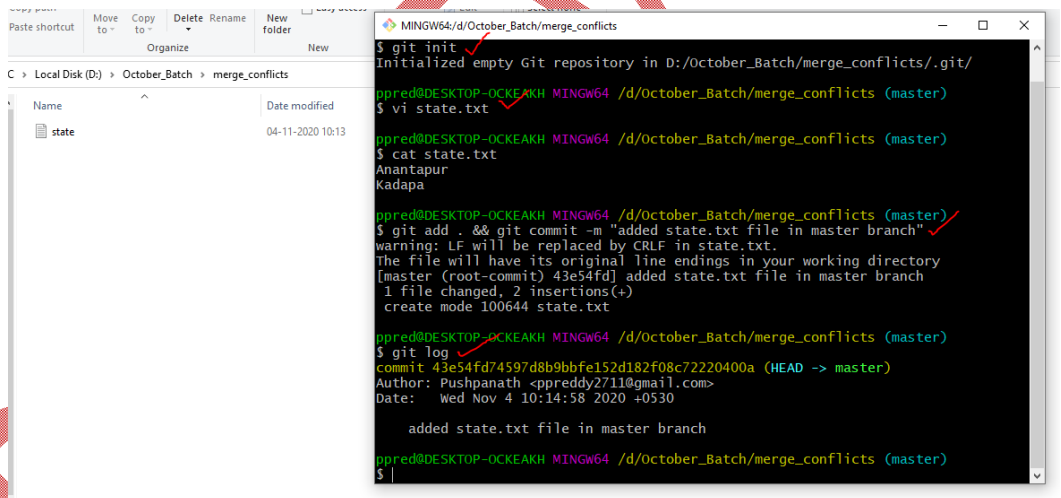
   # git init

   # vi state.txt

   # cat state.txt

   # git add . && git commit -m "added state.txt file in master branch"

   # git log



2. Create dev branch and update the state.txt file

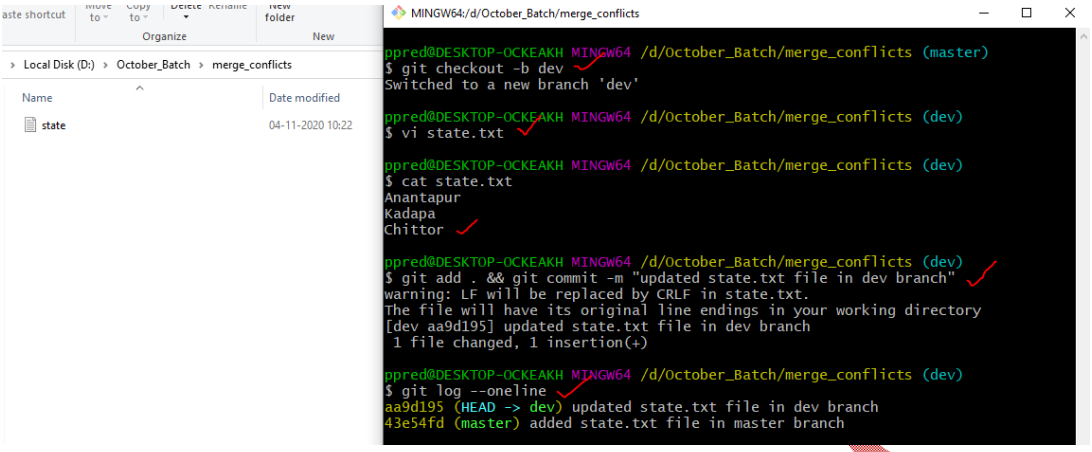
   # git checkout -b dev

   # vi state.txt

   # cat state.txt

   # git add . && git commit -m "updated state.txt file in dev branch"

   # git log --oneline



3. Switch to master branch and update the state.txt file.

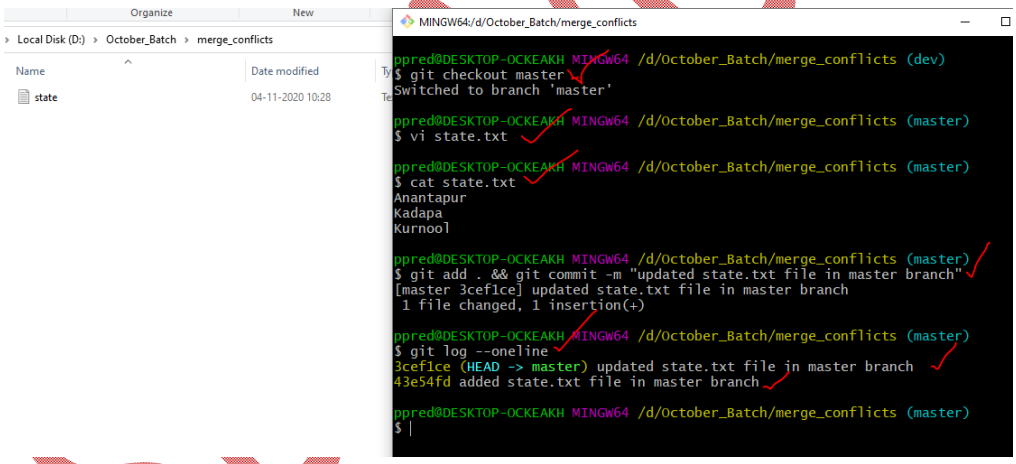
   # git checkout master

   # vi state.txt

   # cat state.txt

   # git add . && git commit -m "updated state.txt file in master branch"

   # git log --oneline



4. Try to merge the dev branch to the master branch.

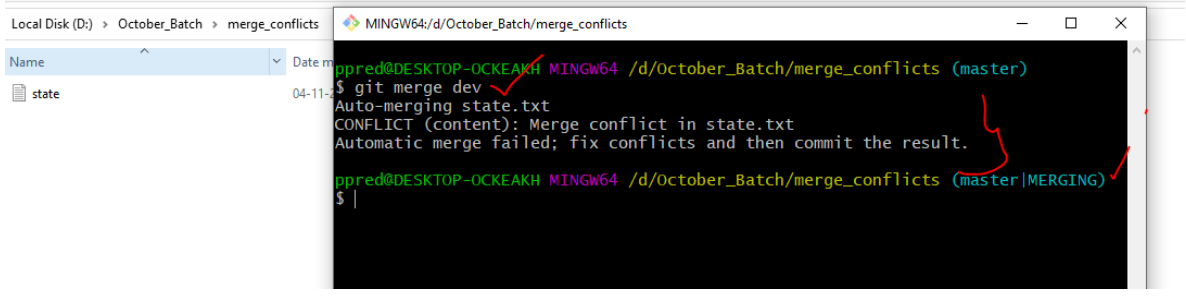
   # git merge dev

   Will get the issue like below

   Auto-merging state.txt

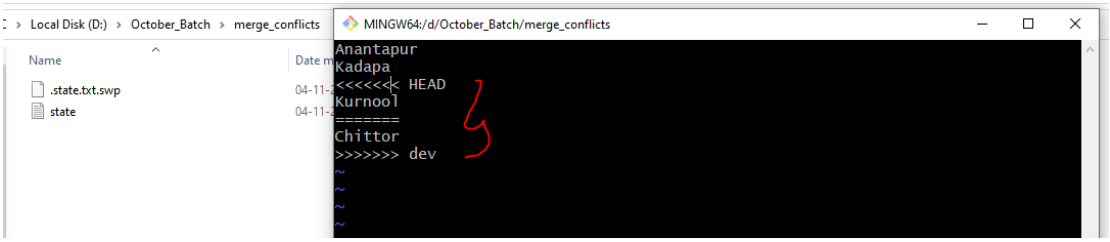
   CONFLICT (content): Merge conflict in state.txt

   Automatic merge failed; fix conflicts and then commit the result.



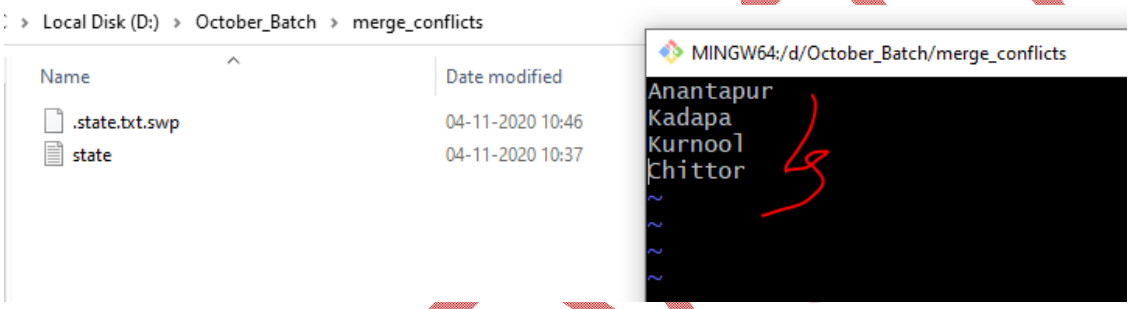
5. Update the state.txt file in master branch.

   # vi state.txt



<<<<<<< HEAD is the current branch i.e. master branch

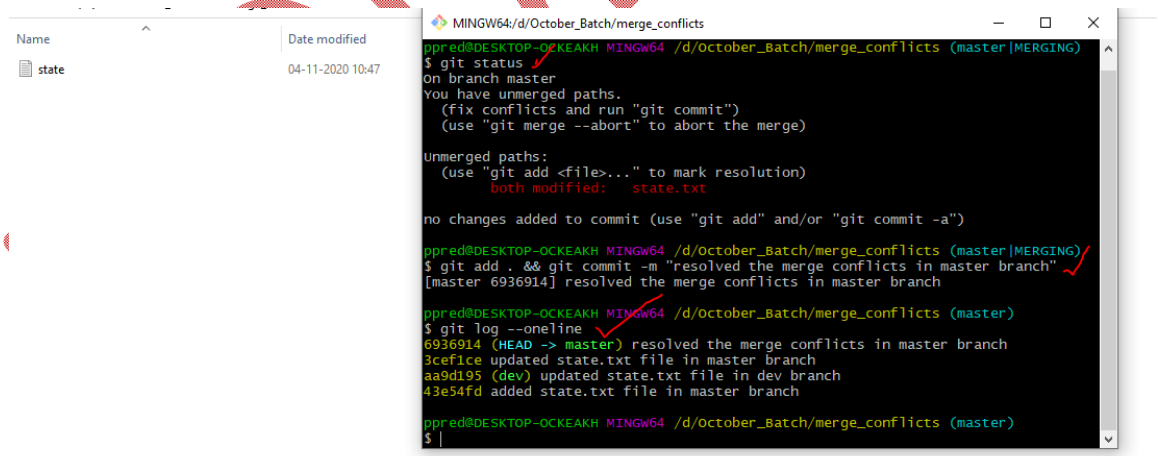
>>>>>>> dev is the remote branch trying to merge.



git status

git add . && git commit -m "resolved the merge conflicts in master branch"

git log --oneline



6. Verify the content in state.txt in master branch.

   # git branch

   # cat state.txt

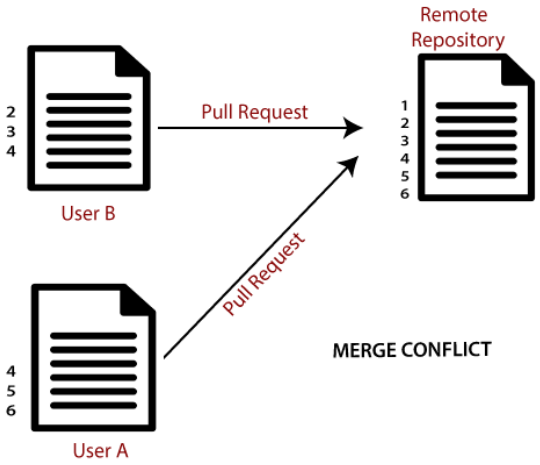
   # git log

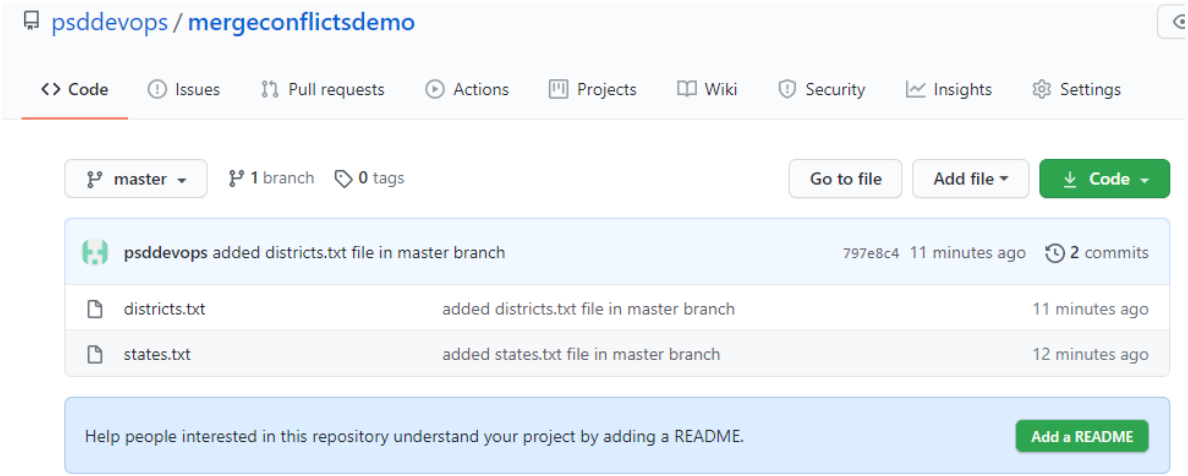
```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (master)
$ git branch
  dev
* master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (master)
$ cat state.txt
Anantapur
Kadapa
Kurnool
Chittor

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (master)
$ git log
commit 693691408253693937787ff361b59fea70dbc1840 (HEAD -> master)
Merge: 3cef1ce aa9d195
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Nov 4 10:51:08 2020 +0530

    resolved the merge conflicts in master branch

commit 3cef1ce7196cdc59048ba9bc184d6174a66a0d45
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Nov 4 10:28:43 2020 +0530

    updated state.txt file in master branch

commit aa9d195f37d1748dfce95d466f904af3a12f28c6 (dev)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Nov 4 10:23:34 2020 +0530

    updated state.txt file in dev branch

commit 43e54fd74597d8b9bbfe152d182f08c72220400a
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Nov 4 10:14:58 2020 +0530

    added state.txt file in master branch

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (master)
$
```

7. Checkout the dev branch and verify the content.

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (master)
$ git checkout dev
Switched to branch 'dev'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (dev)
$ cat state.txt
Anantapur
Kadapa
Chittor

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (dev)
$ git log
commit aa9d195f37d1748dfce95d466f904af3a12f28c6 (HEAD -> dev)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Nov 4 10:23:34 2020 +0530

    updated state.txt file in dev branch

commit 43e54fd74597d8b9bbfe152d182f08c72220400a
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Wed Nov 4 10:14:58 2020 +0530

    added state.txt file in master branch

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (dev)
$
```

Merge conflicts from remote repo





Suppose my remote repository has cloned by two of my team member user1 and user2. The user1 made changes as below in my projects districts.txt file.

git clone https://github.com/psddevops/mergeconflictsdemo.git

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user1/mergeconflictsdemo (master)
$ git clone https://github.com/psddevops/mergeconflictsdemo.git
```

vi districts.txt

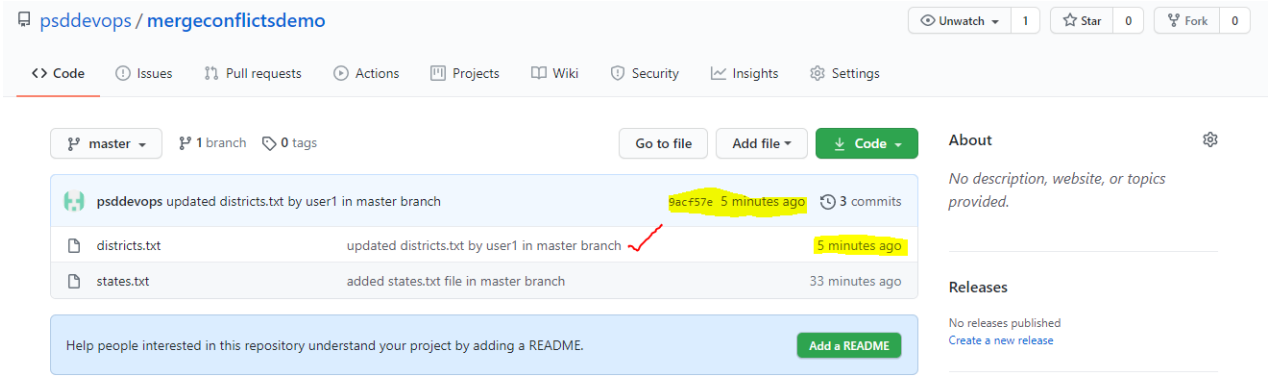
```
Anantapur
Kadapa
Chittor
```

git add . && git commit -m "updated districts.txt by user1 in master branch"

git push origin master

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user1/mergeconflictsdemo (master)
$ ll
total 2
-rw-r--r-- 1 ppred 197609 19 Nov  4 15:49 districts.txt
-rw-r--r-- 1 ppred 197609 27 Nov  4 15:49 states.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user1/mergeconflictsdemo (master)
$ vi districts.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user1/mergeconflictsdemo (master)
$ git add . && git commit -m "updated districts.txt by user1 in master branch"
[master 9acf57e] updated districts.txt by user1 in master branch
 1 file changed, 1 insertion(+)

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user1/mergeconflictsdemo (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 330 bytes | 82.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/psddevops/mergeconflictsdemo.git
   797e8c4..9acf57e  master -> master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user1/mergeconflictsdemo (master)
$
```

Now remote repository is



Now, at the same time, user2 also update the districts.txt file as follows.

vi districts.txt

git add . && git commit -m "updated districts.txt by user2 in master branch"

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user2/mergeconflictsdemo (master)
$ vi districts.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user2/mergeconflictsdemo (master)
$ git add . && git commit -m "updated districts.txt by user2 in master branch"
[master 7bf3100] updated districts.txt by user2 in master branch
 1 file changed, 1 insertion(+)

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user2/mergeconflictsdemo (master)
$ git push origin master
To https://github.com/psddevops/mergeconflictsdemo.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/psddevops/mergeconflictsdemo.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user2/mergeconflictsdemo (master)
$
```

In the above output, the server knows that the file is already updated and not merged with other branches. So, the push request was rejected by the remote server. It will throw an error message like [rejected] failed to push some refs to <remote URL>. It will suggest you to pull the repository first before the push.

git pull origin master

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user2/mergeconflictsdemo (master)
$ git pull origin master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 310 bytes | 1024 bytes/s, done.
From https://github.com/psddevops/mergeconflictsdemo
 * branch            master     -> FETCH_HEAD
   797e8c4..9acf57e  master     -> origin/master
Auto-merging districts.txt
CONFLICT (content): Merge conflict in districts.txt
Automatic merge failed; fix conflicts and then commit the result.

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user2/mergeconflictsdemo (master|MERGING)
$
```

Update the districts.txt file

cat districts.txt

```
Anantapur
Kadapa
<<<<<< HEAD
Kurnool
=======
Chittor
>>>>>>> 9acf57e11b6ca5237cc750a5c2f1d5e967d797ed
~
```

vi districts.txt

```
Anantapur
Kadapa
Kurnool
Chittor
~
~
```

git status

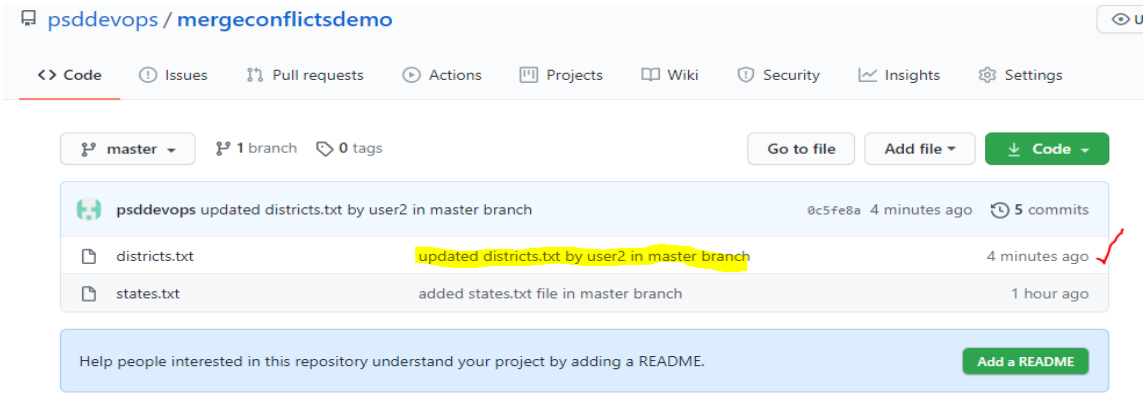
git add . && git commit -m "updated districts.txt by user2 in master branch"

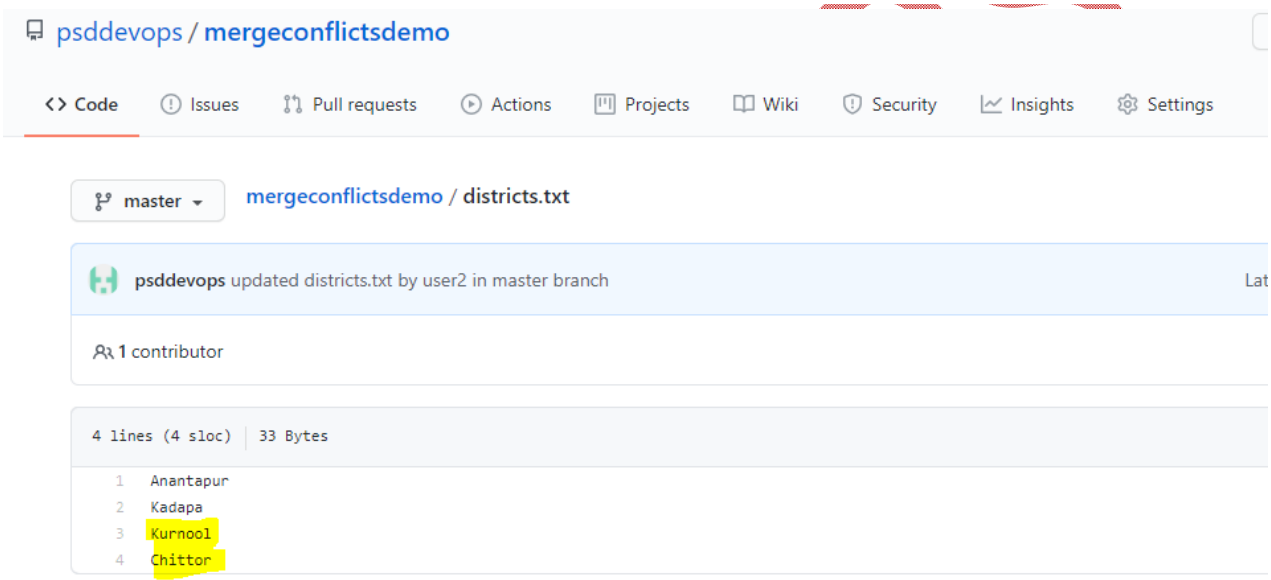
```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user2/mergeconflictsdemo (master|MERGING)
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   districts.txt

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

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user2/mergeconflictsdemo (master|MERGING)
$ git add . && git commit -m "updated districts.txt by user2 in master branch"
[master 0c5fe8a] updated districts.txt by user2 in master branch
```

git push origin master

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user2/mergeconflictsdemo (master)
$ git push origin master
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 663 bytes | 221.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/psddevops/mergeconflictsdemo.git
   9acf57e..0c5fe8a  master -> master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts/user2/mergeconflictsdemo (master)
$
```

Now the remote repository is

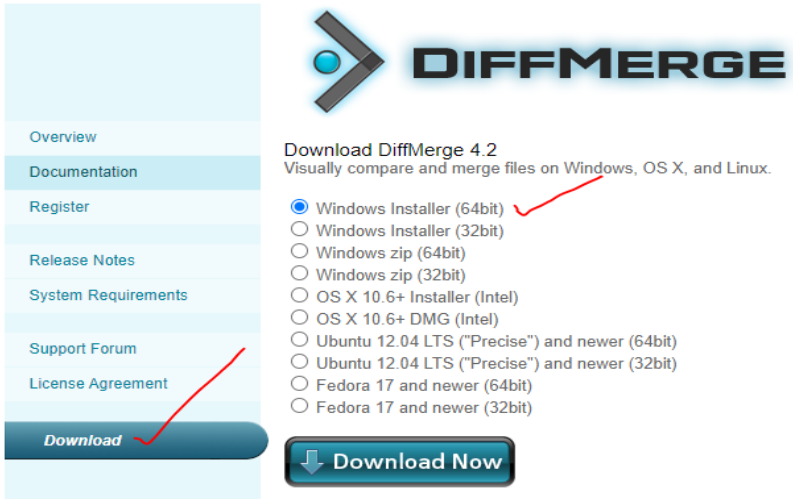




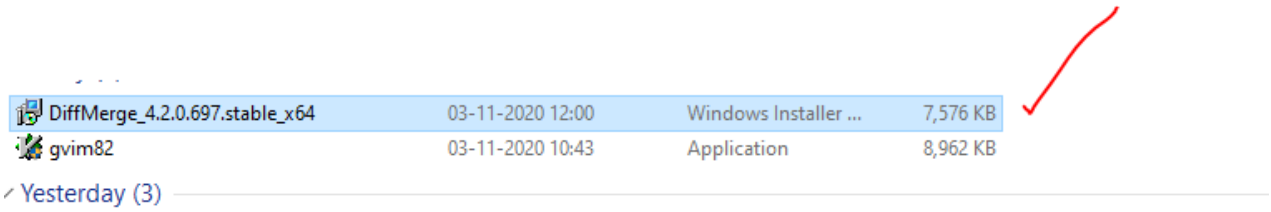
Resolving merge conflicts using merge tools.

1. Downloading diffmerge tool by using below link

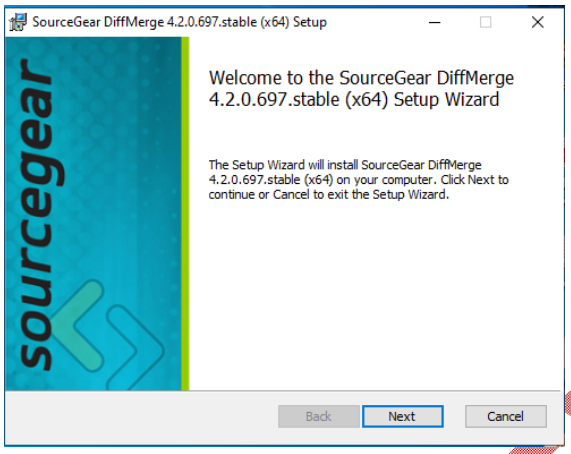
https://sourcegear.com/diffmerge/downloads.php

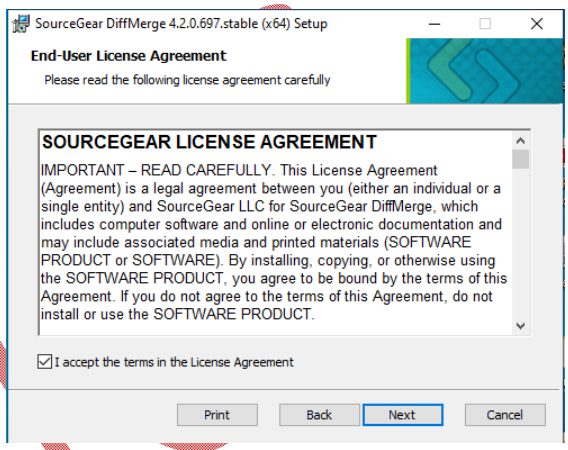


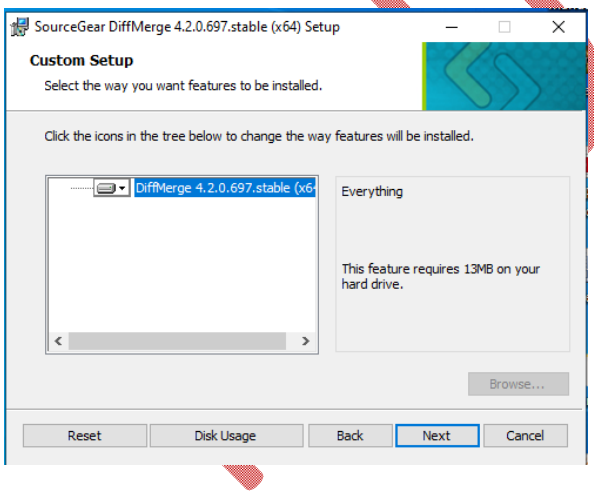
*Prepared by **PPNREDDY** .....*

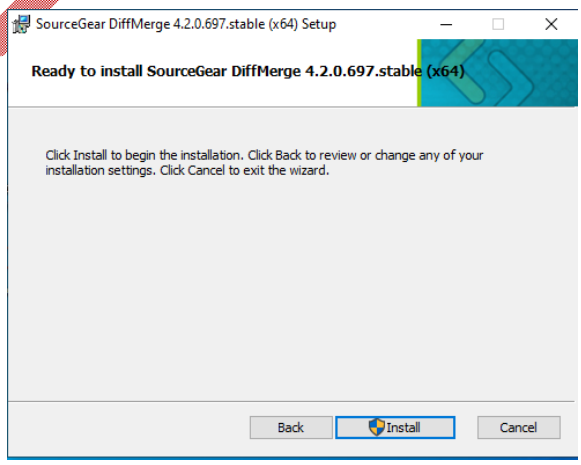


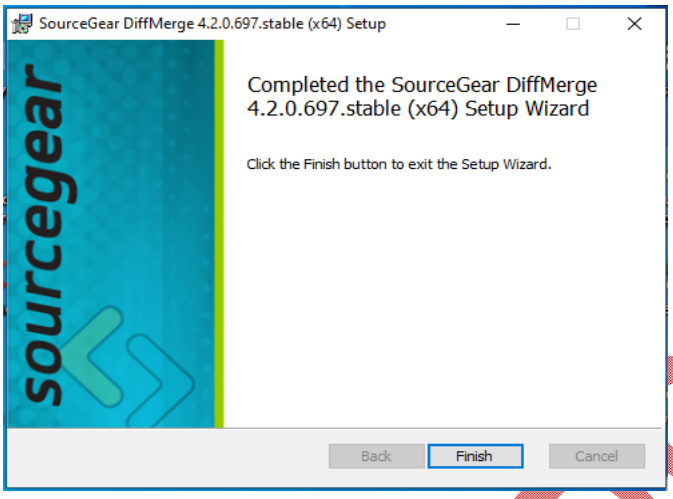
2. Install diffmerge in windows machine.











3. Need to configure vimdiff as a git merge tool.

```
git config --global diff.tool diffmerge
git config --global difftool.diffmerge.cmd 'C:/Program\
Files/SourceGear/Common/DiffMerge/sgdm.exe "$LOCAL" "$REMOTE"'
git config --global merge.tool diffmerge
git config --global mergetool.diffmerge.trustExitCode true
git config --global mergetool.diffmerge.cmd 'C:/Program\
Files/SourceGear/Common/DiffMerge/sgdm.exe -merge -result="$MERGED" "$LOCAL"
"$BASE" "$REMOTE"'
```

```
ppred@DESKTOP-OCKEAKH MINGW64 ~/OneDrive/Desktop
$ git config --global diff.tool diffmerge

ppred@DESKTOP-OCKEAKH MINGW64 ~/OneDrive/Desktop
$ git config --global difftool.diffmerge.cmd 'C:/Program\ Files/SourceGear/Common/DiffMerge/sgdm.exe "$LOCAL" "$REMOTE"'

ppred@DESKTOP-OCKEAKH MINGW64 ~/OneDrive/Desktop
$ git config --global merge.tool diffmerge

ppred@DESKTOP-OCKEAKH MINGW64 ~/OneDrive/Desktop
$ git config --global mergetool.diffmerge.trustExitCode true

ppred@DESKTOP-OCKEAKH MINGW64 ~/OneDrive/Desktop
$ git config --global mergetool.diffmerge.cmd 'C:/Program\ Files/SourceGear/Common/DiffMerge/sgdm.exe -merge -result="$MERGED" "$LOCAL" "$BASE" "$REM
OTE"'

ppred@DESKTOP-OCKEAKH MINGW64 ~/OneDrive/Desktop
$ git config --global --list
user.name=Pushpanath
user.email=ppreddy2711@gmail.com
gui.recentrepo=C:/Users/ppred/OneDrive/Desktop/gitdemo
diff.tool=diffmerge
difftool.diffmerge.cmd=C:/Program\ Files/SourceGear/Common/DiffMerge/sgdm.exe "$LOCAL" "$REMOTE"
merge.tool=diffmerge
mergetool.diffmerge.trustexitcode=true
mergetool.diffmerge.cmd=C:/Program\ Files/SourceGear/Common/DiffMerge/sgdm.exe -merge -result="$MERGED" "$LOCAL" "$BASE" "$REMOTE"

ppred@DESKTOP-OCKEAKH MINGW64 ~/OneDrive/Desktop
$
```

Merge conflicts demo vimdiff tool

1. Create india.txt file in master branch.

   # touch india.txt

   # git init

   # git status

   # git add . && git commit -m "added india.txt file in master branch"

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts
$ git init
Initialized empty Git repository in D:/October_Batch/merge_conflicts/.git/

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (master)
$ git status
On branch master

No commits yet

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

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

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (master)
$ git add . && git commit -m "added india.txt file in master branch"
[master (root-commit) 37a34b6] added india.txt file in master branch
 1 file changed, 4 insertions(+)
 create mode 100644 india.txt
```

2. Create dev branch and update india.txt

   # git checkout -b dev

   # vi india.txt

   # git add . && git commit -m "updated india.txt file in dev branch"

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (master)
$ git checkout -b dev
Switched to a new branch 'dev'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (dev)
$ vi india.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (dev)
$ git add . && git commit -m "updated india.txt file in dev branch"
[dev 9365ca9] updated india.txt file in dev branch
 1 file changed, 1 insertion(+)
```

3. Checkout to master and update the india.txt file

   # git checkout master

   # vi india.txt

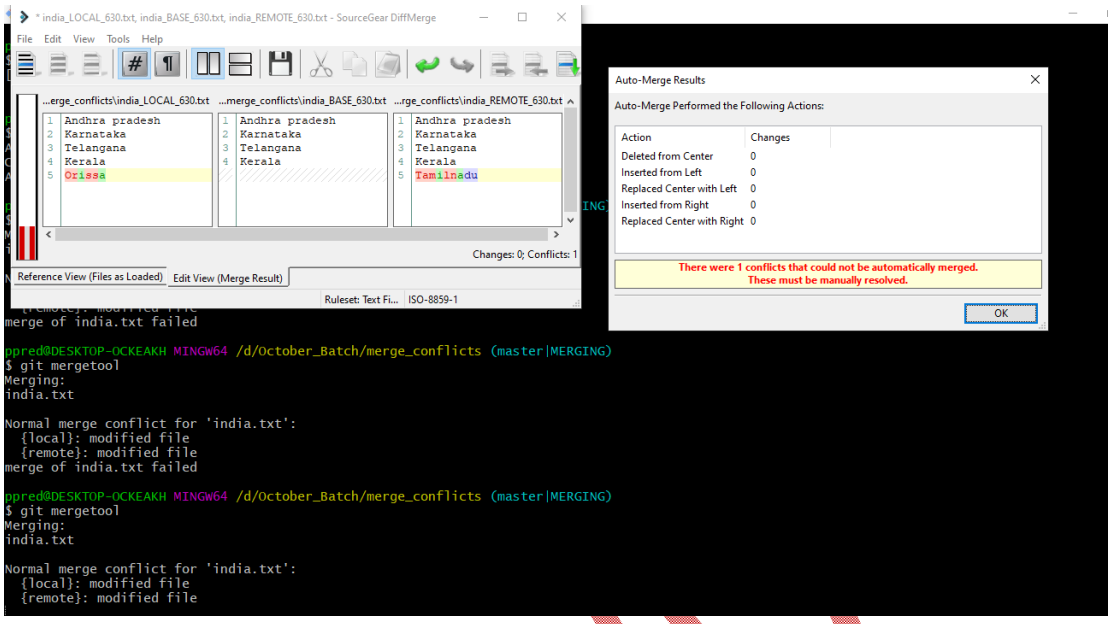
   # git add . && git commit -m "updated india.txt file in master branch"

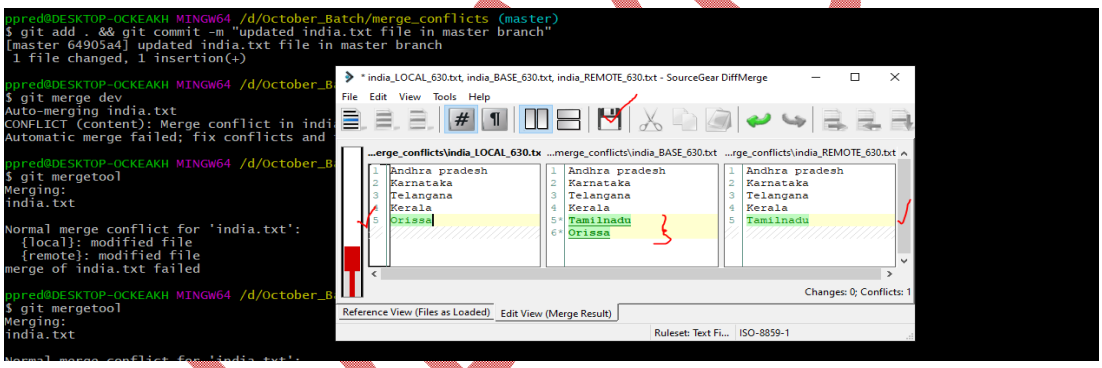
   # git merge dev

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (dev)
$ git checkout master
Switched to branch 'master'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (master)
$ vi india.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (master)
$ git add . && git commit -m "updated india.txt file in master branch"
[master 64905a4] updated india.txt file in master branch
 1 file changed, 1 insertion(+)

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/merge_conflicts (master)
$ git merge dev
Auto-merging india.txt
CONFLICT (content): Merge conflict in india.txt
Automatic merge failed; fix conflicts and then commit the result.
```

4. Try to Merge dev to master branch

   # git mergetool





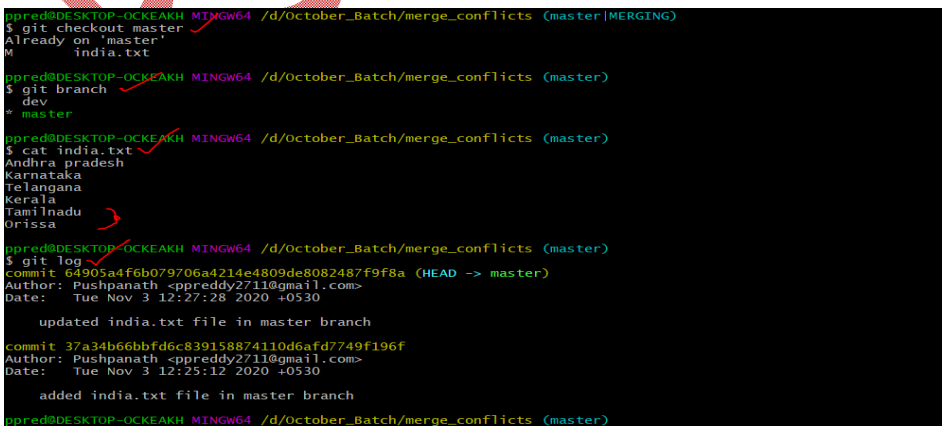
5. Verify the india.txt file in master branch.

   # git checkout master

   # git branch

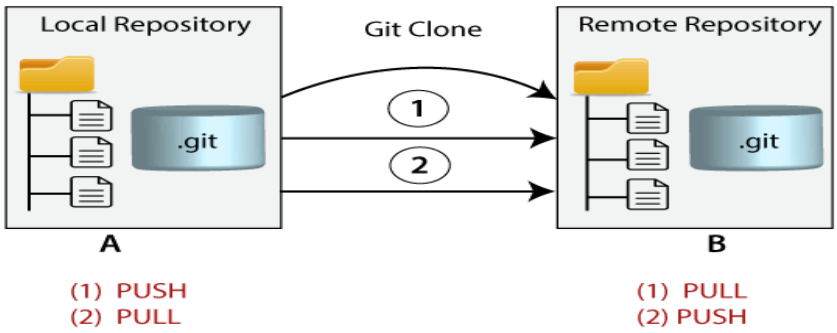
   # cat india.txt

   # git log



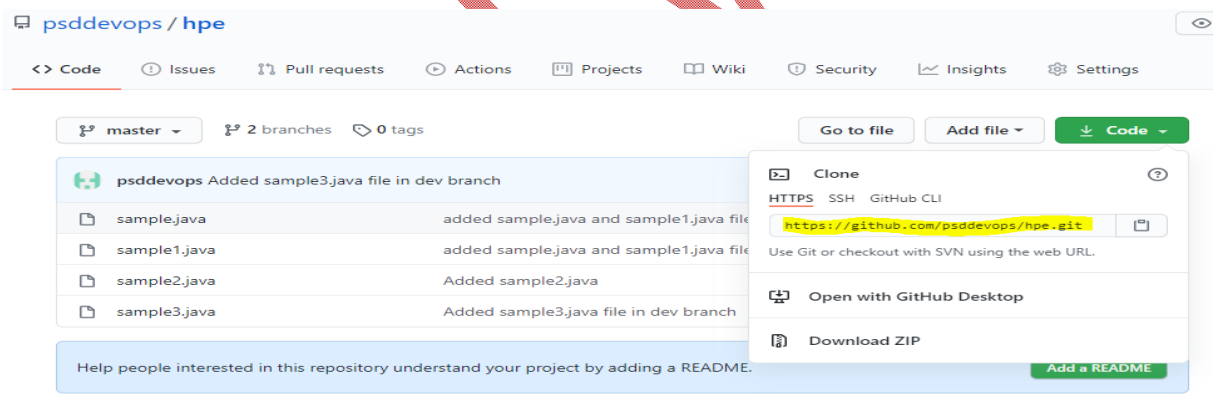
Git clone

In Git, cloning is the act of making a copy of any target repository. The target repository can be remote or local. You can clone your repository from the remote repository to create a local copy on your system. Also, you can sync between the two locations.



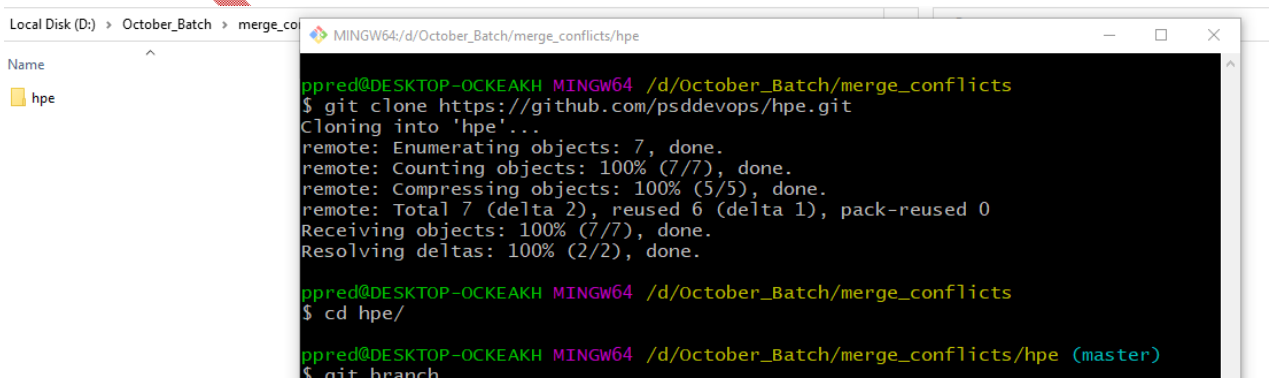
The git clone is a command-line utility which is used to make a local copy of a remote repository. It accesses the repository through a remote URL. Usually, the original repository is located on a remote server, often from a Git service like GitHub, Bitbucket, or GitLab. The remote repository URL is referred to the origin.

Cloning a repository from Github

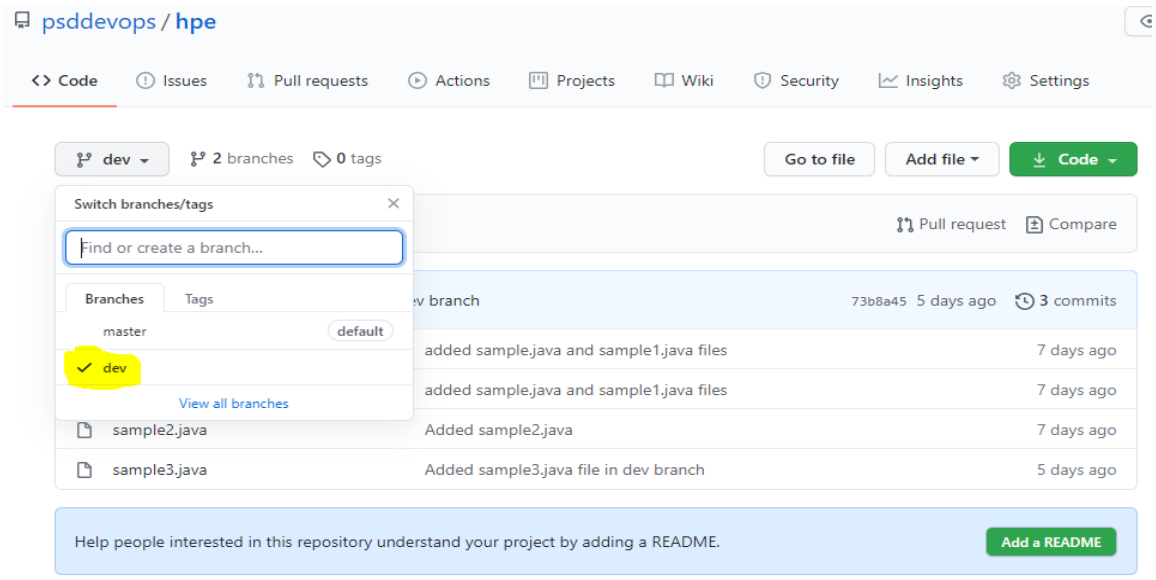


Copy the URL (https://github.com/psddevops/hpe.git)

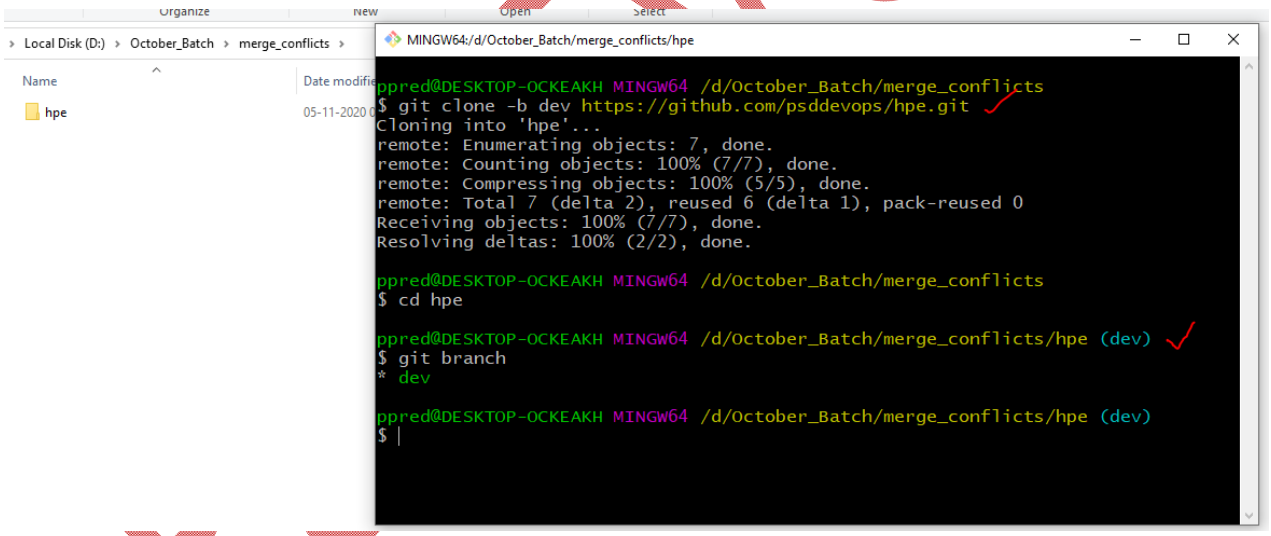
git clone https://github.com/psddevops/hpe.git



Cloning a specific branch from git repository



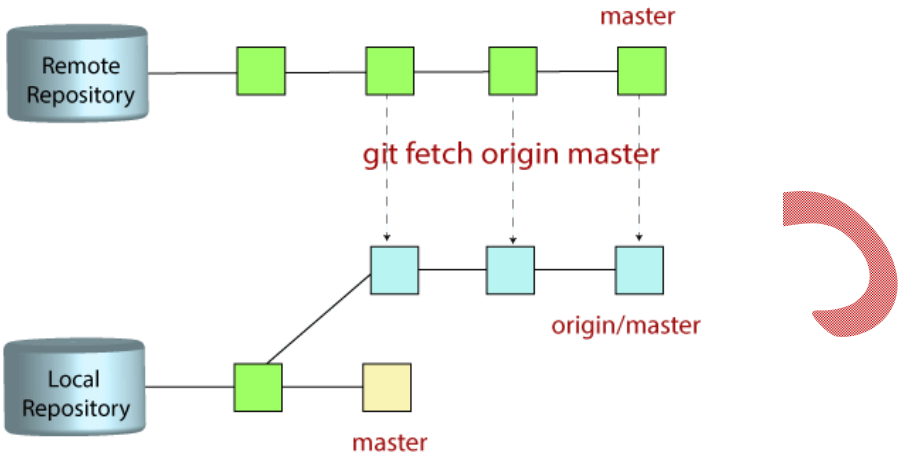
git clone -b dev https://github.com/psddevops/hpe.git



*Prepared by **PPNREDDY** .....*

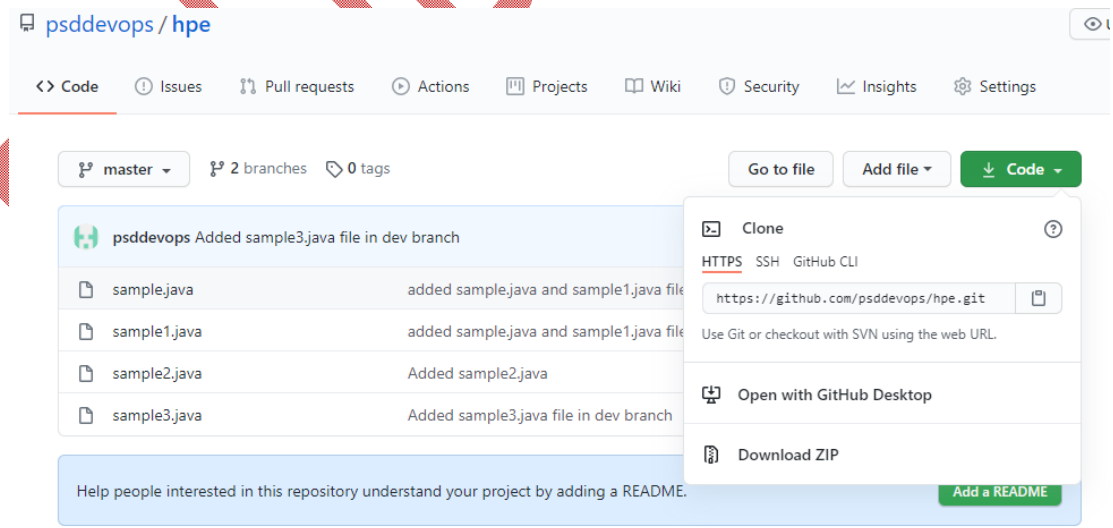
GIT Fetch

Git "fetch" Downloads commits, objects and refs from another repository. It fetches branches and tags from one or more repositories. It holds repositories along with the objects that are necessary to complete their histories to keep updated remote-tracking branches.



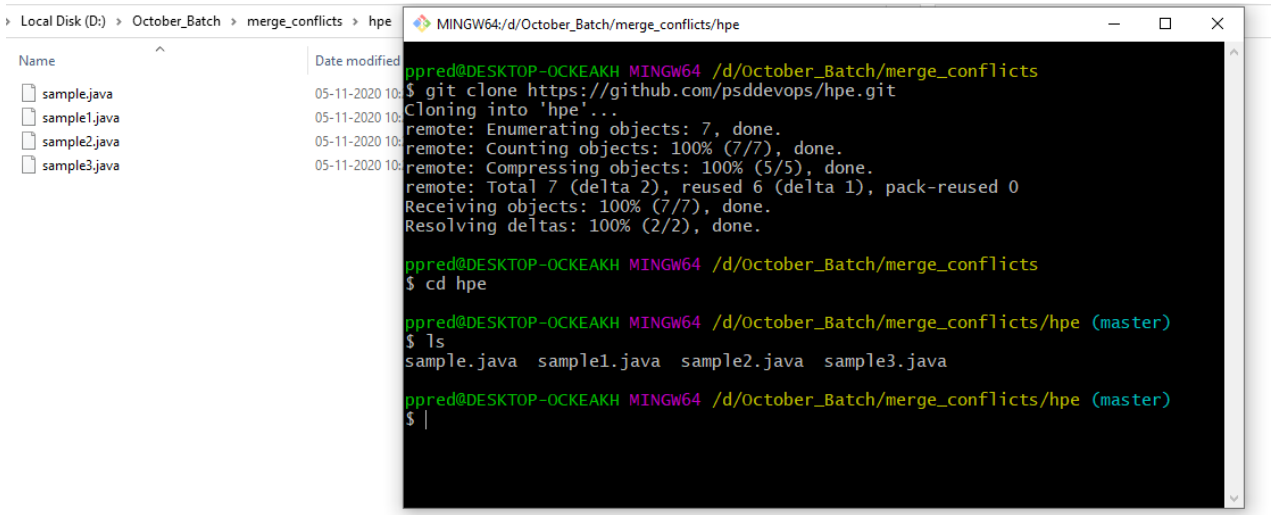
The "**git fetch**" command is used to pull the updates from remote-tracking branches. Additionally, we can get the updates that have been pushed to our remote branches to our local machines without merging. As we know, a branch is a variation of our repositories main code, so the remote-tracking branches are branches that have been set up to pull and push from remote repository.

Remote repository

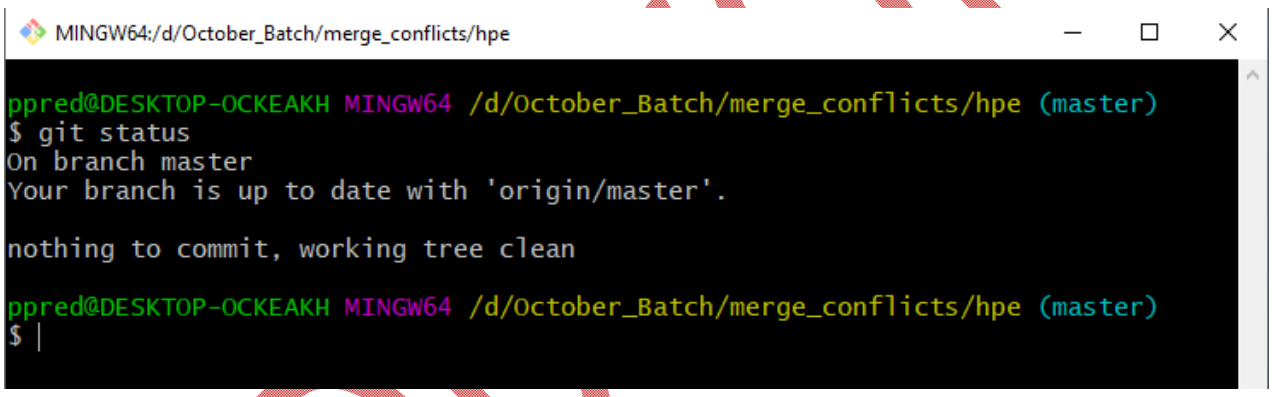


Cloning to the git repository

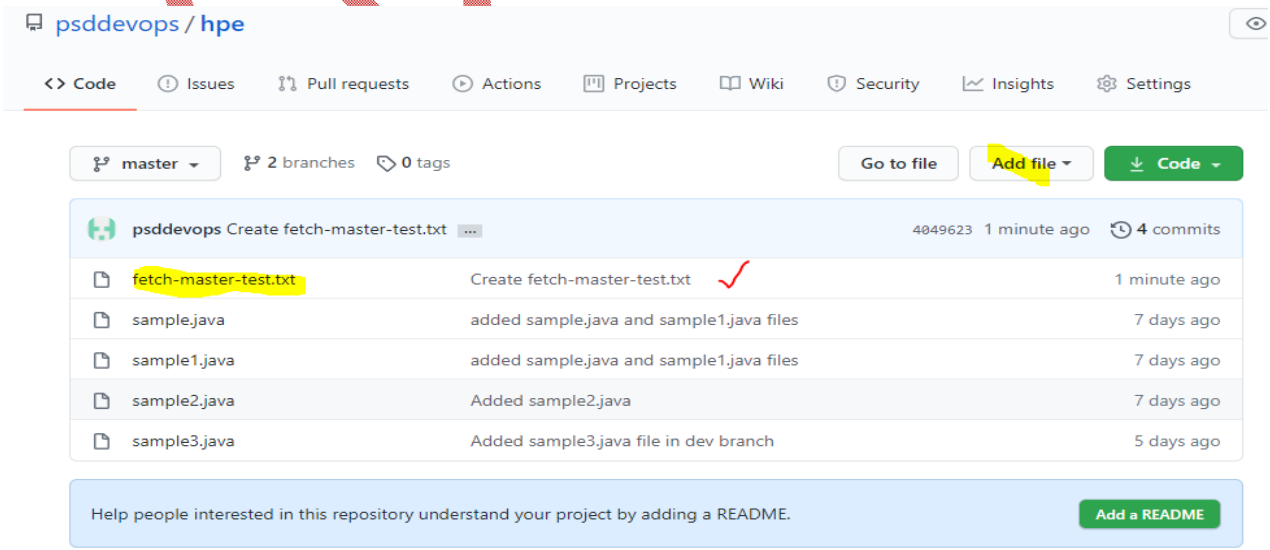
git clone https://github.com/psddevops/hpe.git



git status

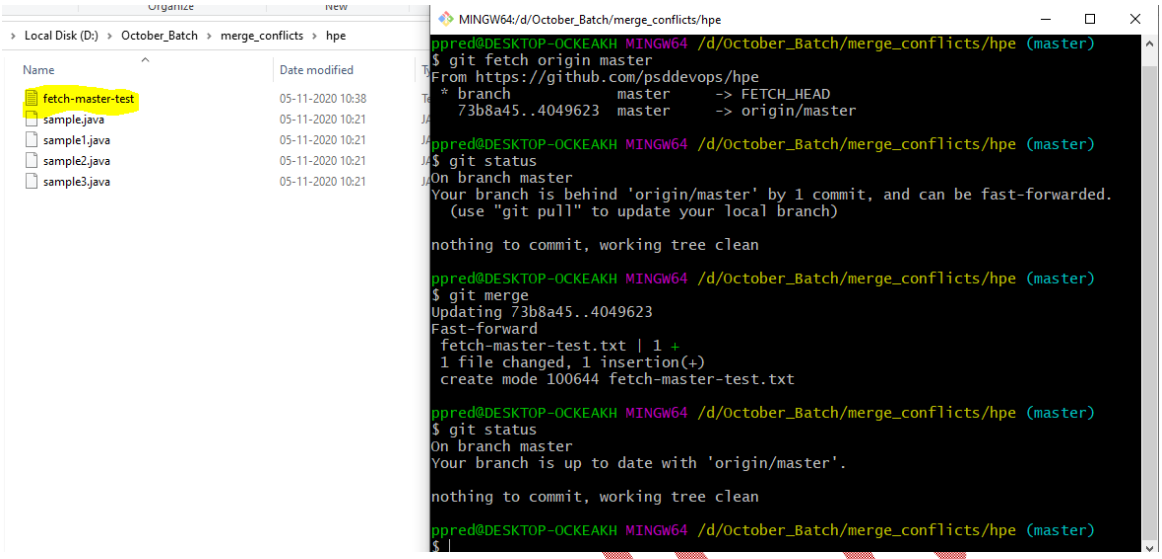


Added a file to the remote repo



*Prepared by **PPNREDDY** .....*

git fetch origin master
git status
git merge



Note:

If we use git fetch and compulsory will run git merge command.

1. To fetch a specific branch
   git fetch <branch URL><branch name>
2. To fetch all the branches simultaneously.
   git fetch –all
3. To synchronize the local repository
   git fetch origin master

GIT PULL

Git pull command is used to fetch the files from remote repository to local repo. If you want to push any commits you must be in synchronize with remote git repository. Synchronize is nothing but whatever the files and commits in remote github repository must be in local repository. Then only you can push commits to remote repo otherwise it will through you errors. To synchronize local repository with remote github repository we use pull command.

What exactly does the git pull command

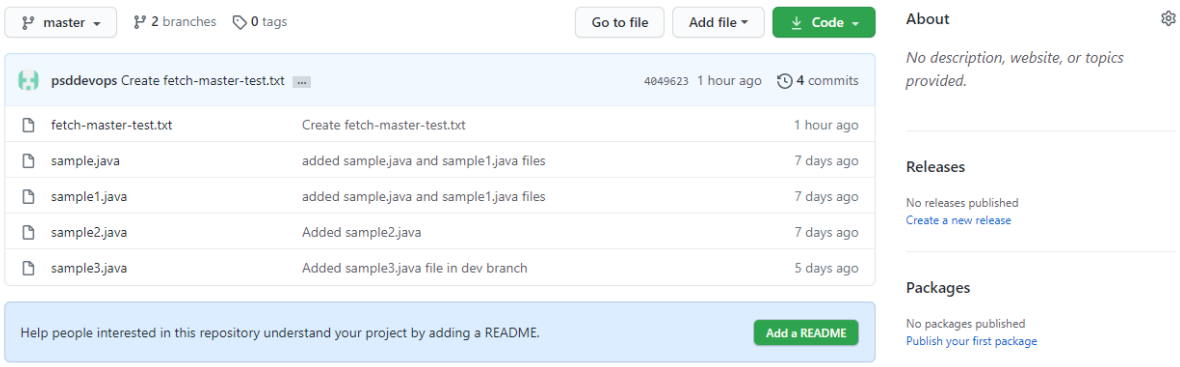
When you are executing pull command it will execute two commands in background.

git fetch + git merge

Fetch command will download all files from remote repo to local repo and Fetch doesn't change local working copies or local files. Merge command will merge the files , so finally you will get the files from remote repo to local repo.

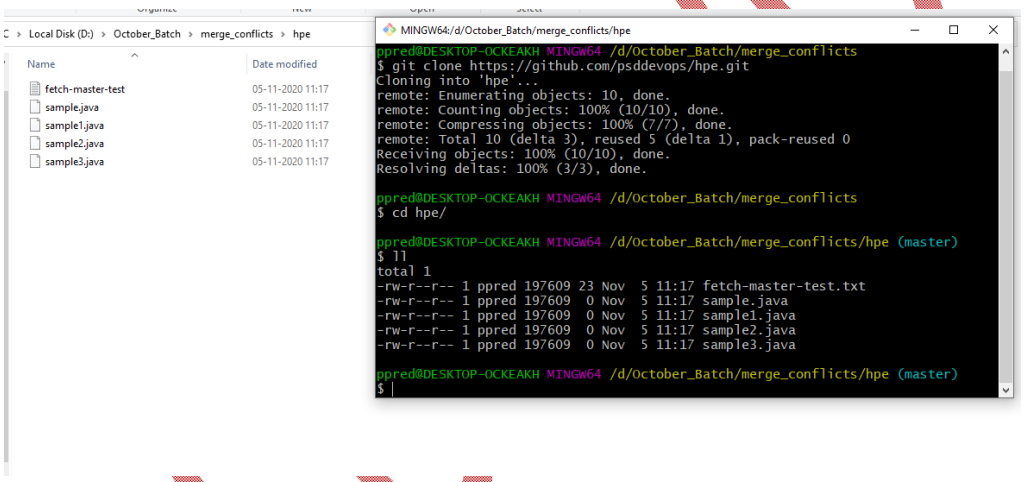
Git Pull Demo

Git hub remote repository

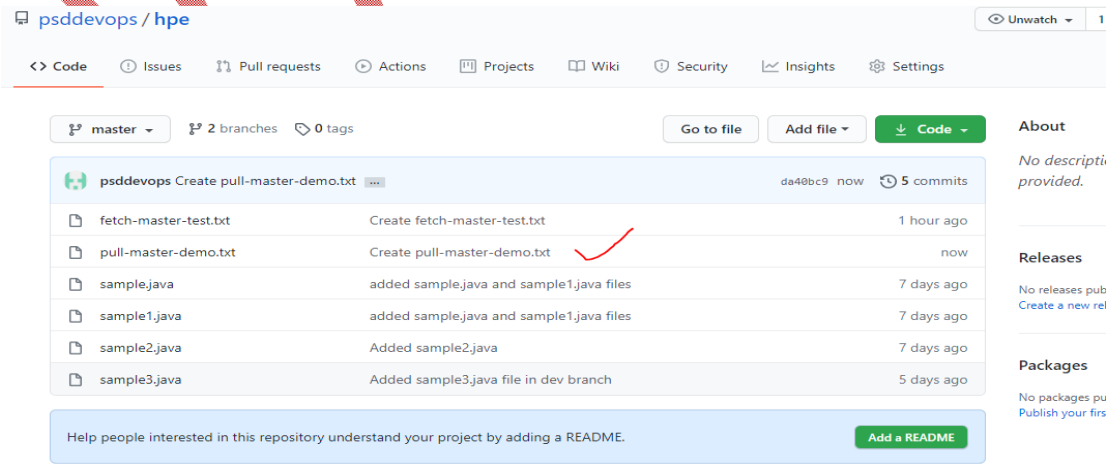


Clone the repository to local

git clone https://github.com/psddevops/hpe.git



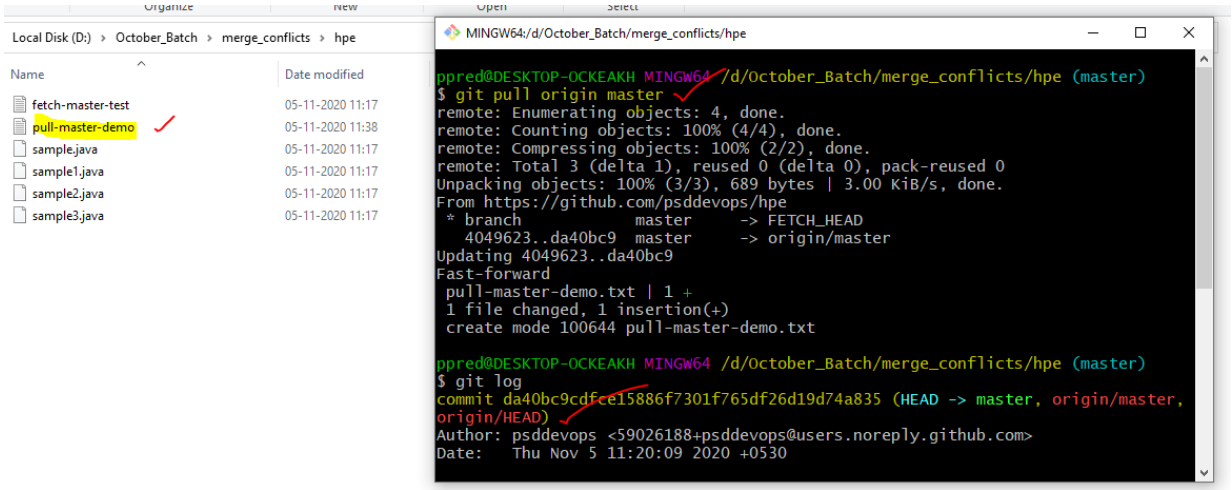
Adding a file in remote repo



*Prepared by **PPNREDDY** .....*

Pull the git repository

git pull origin master



Git diff

Git diff is a command-line utility. It's a multiuse Git command. When it is executed, it runs a diff function on Git data sources. These data sources can be files, branches, commits, and more. It is used to show changes between commits, commit, and working tree, etc.

It compares the different versions of data sources. The version control system stands for working with a modified version of files. So, the diff command is a useful tool for working with Git.

Track the changes that have not been staged

cat test.java

vi test.java

git diff

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ cat test.java
This is first line

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ vi test.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git diff
warning: LF will be replaced by CRLF in test.java.
The file will have its original line endings in your working directory
diff --git a/test.java b/test.java
index 0cbcf32..655a706 100644
--- a/test.java
+++ b/test.java
@@ -1 +1,2 @@
 This is first line
+This is second line

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ cat test.java
This is first line
This is second line

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Track the changes that have staged but not committed

cat test.java

git add .

git diff –staged

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ cat test.java
This is first line
This is second line

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git add .
warning: LF will be replaced by CRLF in test.java.
The file will have its original line endings in your working directory

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git diff --staged
diff --git a/test.java b/test.java
index 0cbcf32..655a706 100644
--- a/test.java
+++ b/test.java
@@ -1 +1,2 @@
 This is first line
+This is second line

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Track the changes after committing a file

git status

git commit -m "updated test.java file"

git diff 5ac0042 62c1d6c

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   test.java


ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git commit -m "updated test.java file"
[master 5ac0042] updated test.java file
 1 file changed, 1 insertion(+)

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log
commit 5ac0042de87052e77741ebf33b34d7a9d9d1cc25 (HEAD -> master)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Thu Nov 5 17:22:51 2020 +0530

    updated test.java file

commit 62c1d6c92b0249509020c971ef7ff7098397bde2
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Thu Nov 5 17:13:23 2020 +0530

    added test.java file

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git diff HEAD

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git diff 5ac0042 62c1d6c
diff --git a/test.java b/test.java
index 655a706..0cbcf32 100644
--- a/test.java
+++ b/test.java
@@ -1,2 +1 @@
 This is first line
-This is second line

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Git Diff Branches

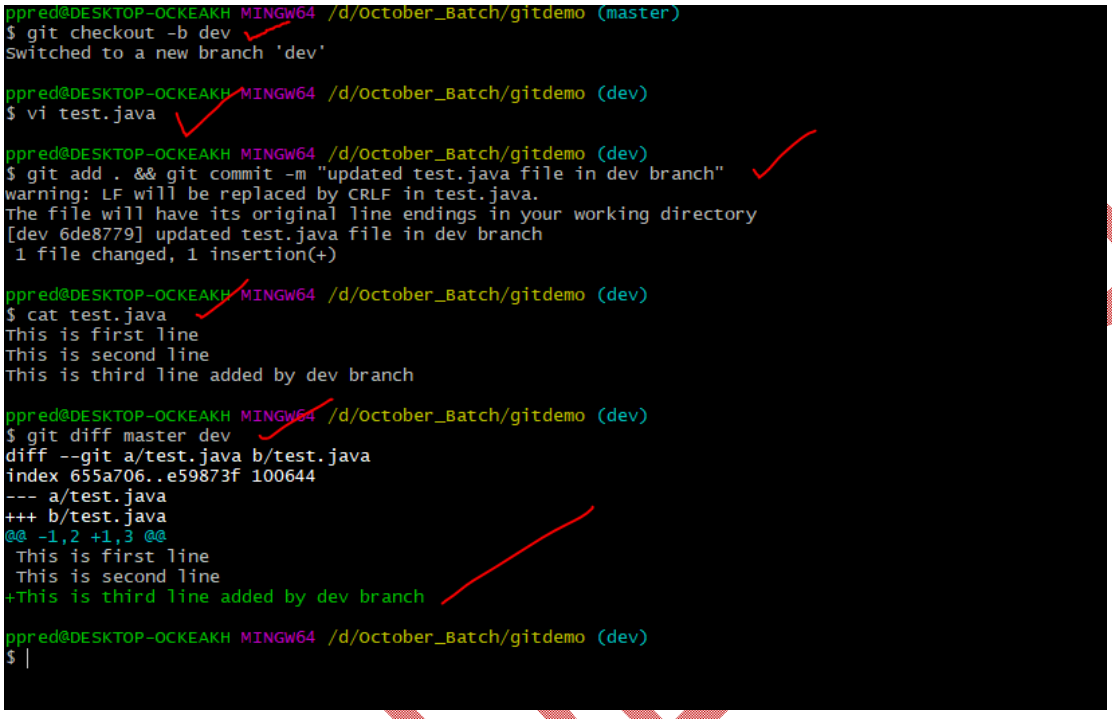
git checkout -b dev

vi test.java

git add . && git commit -m "updated test.java file in dev branch"

cat test.java

git diff master dev



Renaming the branches in local

By using git branch command we can rename git branch

Syntax

Checkout to current branch

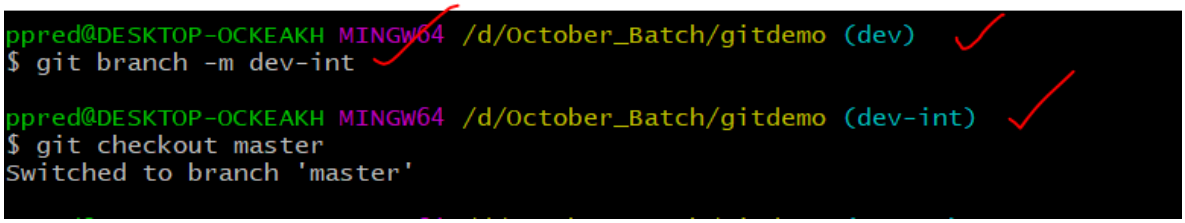
git branch –m <new-branch-name>

From different branch

git branch –m <old-branch-name> <new-branch-name>

git checkout dev

git branch -m dev-int



git branch

git branch -m dev-int uat

git branch

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch
  dev
  dev-int
* master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch -m dev-int uat

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch
  dev
* master
  uat

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Creating a branch from commit-id

Syntax

git branch <branch _name> <commit id>

git log

git branch

git branch commit-branch 5ac0042de87052e77741ebf33b34d7a9d9d1cc25

git branch

git checkout commit-branch

git log –oneline

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log
commit 5ac0042de87052e77741ebf33b34d7a9d9d1cc25 (HEAD -> master)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Thu Nov 5 17:22:51 2020 +0530

    updated test.java file

commit 62c1d6c92b0249509020c971ef7ff7098397bde2
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Thu Nov 5 17:13:23 2020 +0530

    added test.java file

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch
  dev
* master
  uat

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch commit-branch 5ac0042de87052e77741ebf33b34d7a9d9d1cc25

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git branch
  commit-branch
  dev
* master
  uat

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git checkout commit-branch
Switched to branch 'commit-branch'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (commit-branch)
$ git log --oneline
5ac0042 (HEAD -> commit-branch, master) updated test.java file
62c1d6c added test.java file

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (commit-branch)
$
```

Git Stash

Sometimes you want to switch the branches, but you are working on an incomplete part of your current project. You don't want to make a commit of half-done work. Git stashing allows you to do so. The *git stash* command enables you to switch branches without committing the current branch. Git stash temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then come back and reapply them later on. . Stashing is handy if you need to quickly switch context and work on something else, but you're mid-way through a code change and aren't quite ready to commit.

It will try to backup all your changes before you commit. It will do undo the changes.

Demo

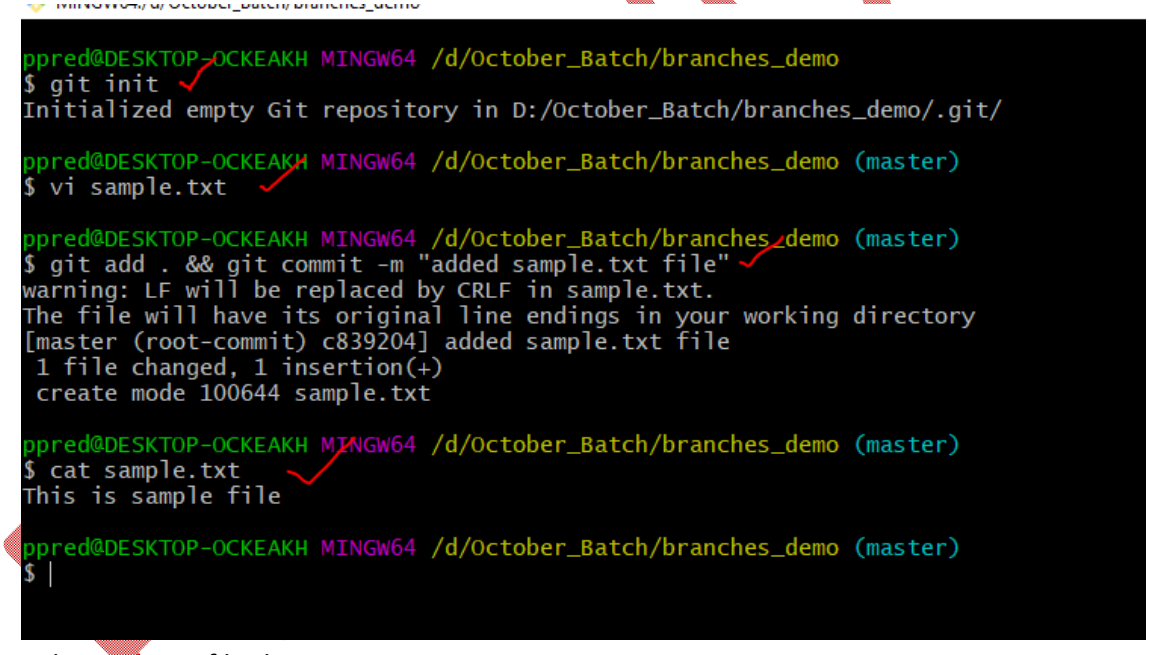
Add sample.txt file with content

git init

vi sample.txt

git add . && git commit -m "added sample.txt file"

cat sample.txt



Updated sample.txt file three times

vi sample.txt

git status

git stash save "added idea-1"

git status

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ cat sample.txt
This is sample file

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ vi sample.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git status
On branch master
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:   sample.txt

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

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash save "added idea-1"
warning: LF will be replaced by CRLF in sample.txt.
The file will have its original line endings in your working directory
Saved working directory and index state On master: added idea-1

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git status
On branch master
nothing to commit, working tree clean

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$
```

vi sample.txt

cat sample.txt

git stash save "added idea-2"

git stash list

git status

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ vi sample.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ cat sample.txt
This is sample file
This is idea-2


ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash save "added idea-2"
Saved working directory and index state On master: added idea-2

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash list
stash@{0}: On master: added idea-2
stash@{1}: On master: added idea-1

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git status
On branch master
nothing to commit, working tree clean

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$
```

vi sample.txt

cat sample.txt

git stash save "added idea-3"

git stash list

git status

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ vi sample.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ cat sample.txt
This is sample file
This is idea-3

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash save "added idea-3"
Saved working directory and index state On master: added idea-3

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash list
stash@{0}: On master: added idea-3
stash@{1}: On master: added idea-2
stash@{2}: On master: added idea-1

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git status
On branch master
nothing to commit, working tree clean

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$
```

git stash show : This command shows the summary of the stash diffs. This command considers only the latest stash.

git stash show -p : It will give you the detailed list of differences.

git stash list: To display list of stashes

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash show
 sample.txt | 1 +
 1 file changed, 1 insertion(+)

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash show -p
diff --git a/sample.txt b/sample.txt
index 175aac5..069e553 100644
--- a/sample.txt
+++ b/sample.txt
@@ -1 +1,2 @@
 This is sample file
+This is idea-3

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash list
stash@{0}: On master: added idea-3
stash@{1}: On master: added idea-2
stash@{2}: On master: added idea-1

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$
```

git stash pop: It apply the latest stash and then immediately drop it from your stack.

git stash pop stash@{1} : It apply the particular stash and then immediately drop it from your stack.

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash list
stash@{0}: On master: added idea-3
stash@{1}: On master: added idea-2
stash@{2}: On master: added idea-1

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash pop
On branch master
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:   sample.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (834e40ce77da6b9608e27dce85765c4a7a4e3461)

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash list
stash@{0}: On master: added idea-2
stash@{1}: On master: added idea-1

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$
```

git stash apply: It copies changes stashed to working/index area, entry still remains in the stash

git stash clear: Remove the stash entries.

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash list
stash@{0}: On master: this is idea-2
stash@{1}: On master: this is idea-1

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash apply stash@{0}
On branch master
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:   sample.txt

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

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$
```

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash list
stash@{0}: On master: this is idea-2
stash@{1}: On master: this is idea-1

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash clear

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$ git stash list

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/branches_demo (master)
$
```

GIT Tags

A name which is attached to the commit-id is called tag. Later point of time we can use tags rather than commit-id. It's a pointer pointing to a specific commit, tag is similar to branch but, branch is used for development/bugfix, whereas tag is used for releasing software. On a branch we can do commits, but on a tag we cannot perform commits.

- We can't attach one tag to multiple/other commit-ids.
- One commit-id having multiple tags.
- One tag having only one commit-id.

Creating a tag

git tag -a VERSION-1.0 -m "Created version 1.0" 4049623

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git log --oneline
da40bc9 (HEAD -> master, origin/master, origin/HEAD) Create pull-master-demo.txt
4049623 Create fetch-master-test.txt
73b8a45 (origin/dev) Added sample3.java file in dev branch
3b8213c Added sample2.java
7e54aa8 added sample.java and sample1.java files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git tag -a VERSION-1.0 -m "Created version 1.0" 4049623

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
```

git tag → list of tags

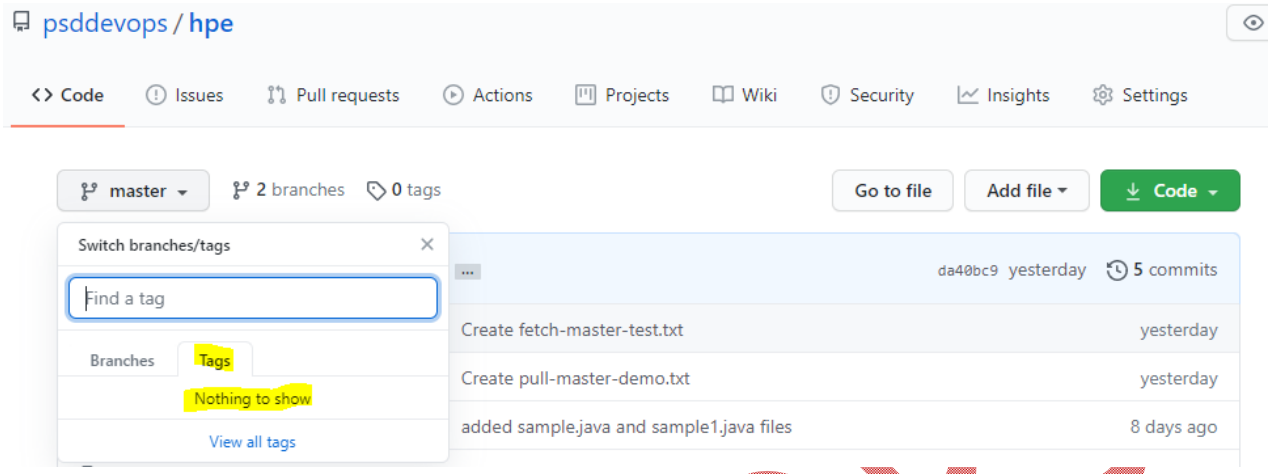
```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git tag -a VERSION-1.1 -m "Created version 1.1" da40bc9

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git tag
VERSION-1.0
VERSION-1.1
```

git show VERSION-1.0 → Displaying the tag details

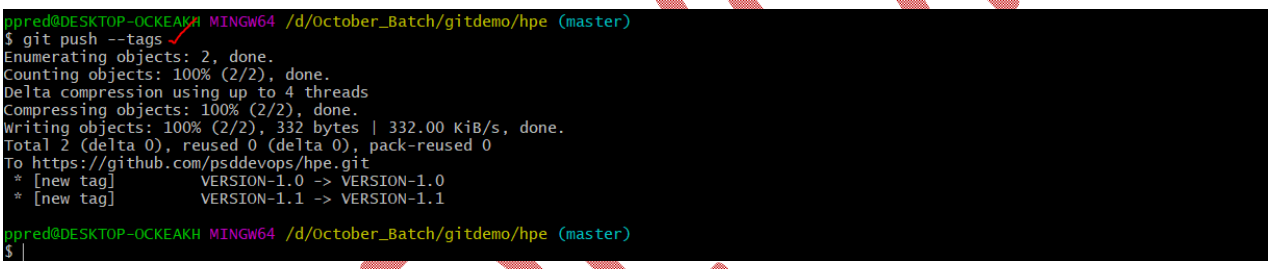
```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git show VERSION-1.0
tag VERSION-1.0
Tagger: Pushpanath <ppreddy2711@gmail.com>
Date:   Fri Nov 6 17:14:47 2020 +0530

Created version 1.0

commit 4049623cfd14a7decb727d26a0affebe5452e6e9 (tag: VERSION-1.0)
Author: psddevops <59026188+psddevops@users.noreply.github.com>
Date:   Thu Nov 5 10:29:46 2020 +0530

    Create fetch-master-test.txt

    fetch-master-test.txt

diff --git a/fetch-master-test.txt b/fetch-master-test.txt
new file mode 100644
index 0000000..16140b1
--- /dev/null
+++ b/fetch-master-test.txt
@@ -0,0 +1 @@
+fetch-master-test.txt
```

Pushing Tags to remote repo

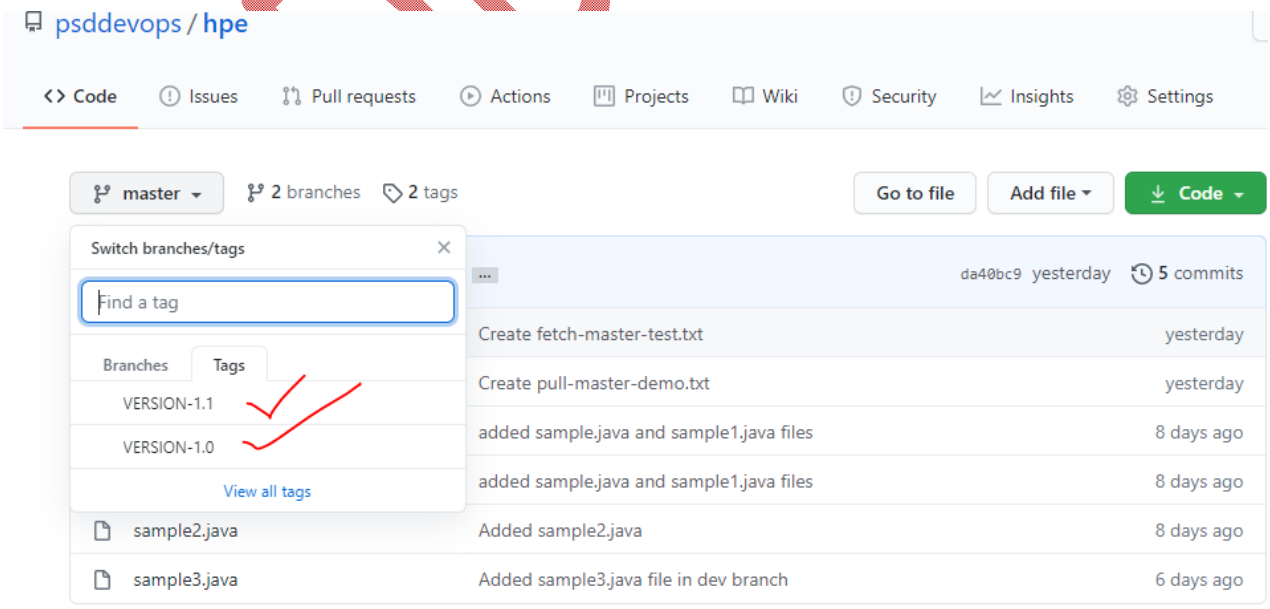
Zero tags in remote repo



git push –tags



Now remote repo



Deleting a tag

git tag -d VERSION-0.1

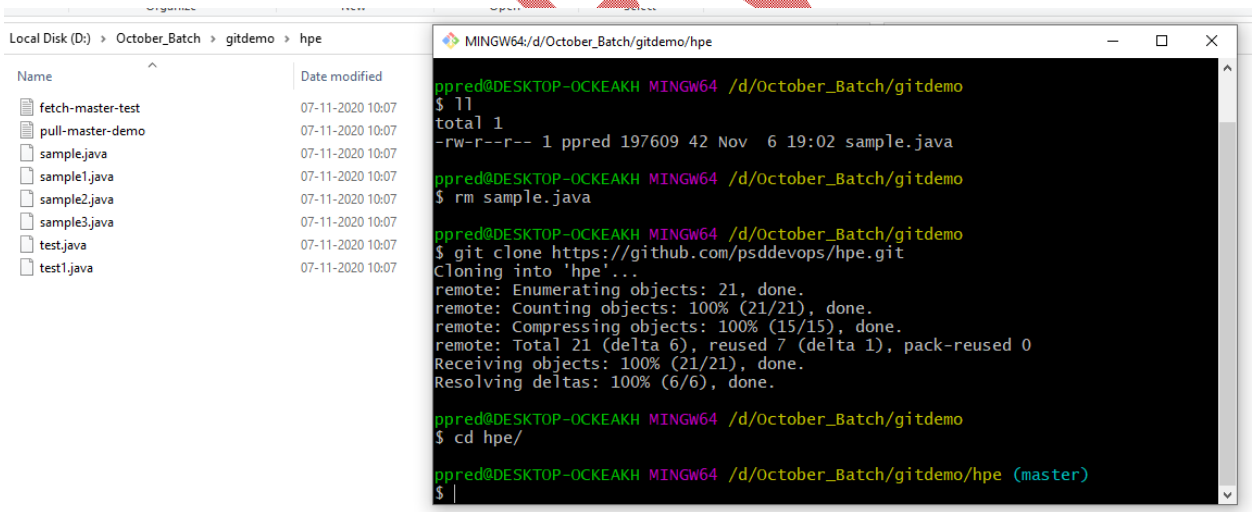
```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git log --oneline
da40bc9 (HEAD -> master, tag: VERSION-1.1, origin/master, origin/HEAD) Create pull-master-demo.txt
4049623 (tag: VERSION-1.0) Create fetch-master-test.txt
73b8a45 (origin/dev) Added sample3.java file in dev branch
3b8213c (tag: VERSION-0.1) Added sample2.java
7e54aa8 added sample.java and sample1.java files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git tag -d VERSION-0.1
Deleted tag 'VERSION-0.1' (was 3047417)

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git log --oneline
da40bc9 (HEAD -> master, tag: VERSION-1.1, origin/master, origin/HEAD) Create pull-master-demo.txt
4049623 (tag: VERSION-1.0) Create fetch-master-test.txt
73b8a45 (origin/dev) Added sample3.java file in dev branch
3b8213c Added sample2.java
7e54aa8 added sample.java and sample1.java files

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$
```

Ignore Files in Git (git .ignore)

Sometimes we don't want to commit the files, which are generated by IDE like *.project, folders* and *.classpath* files or some node module folders like node_module folder into a git repository. To ignore these files and folders to commit we will create one file called .gitignore and we will keep the file names or directory names which we don't want to commit as follows.

Ignoring .class, .conf, .classpath and newfolder files

git clone https://github.com/psddevops/hpe.git



Create new files with below extensions

sample.class
test.class
sample.conf
test.conf
.classpath
.project
newfolder

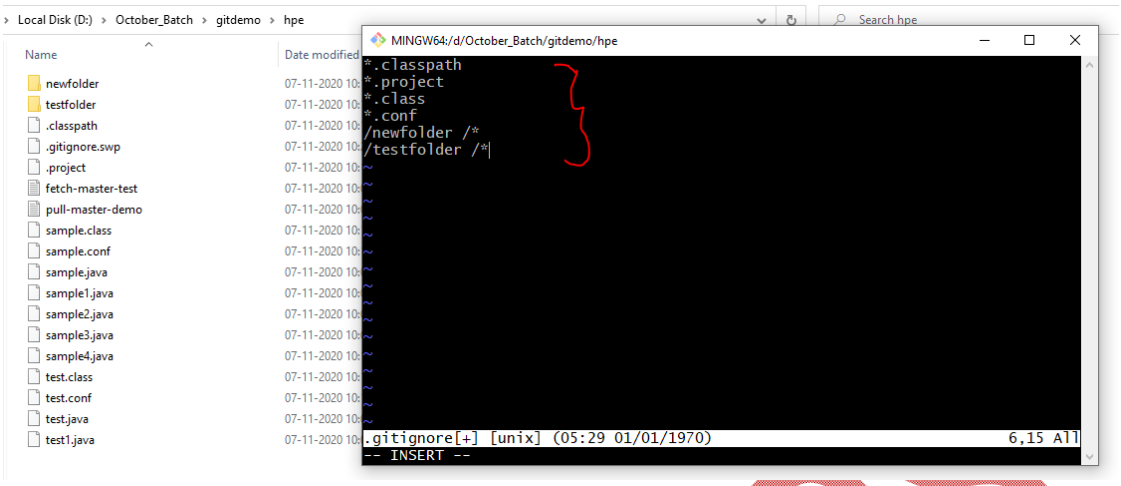
*Prepared by **PPNREDDY** .....*

cd hpe
touch sample.class test.class sample.conf test.conf .classpath .project
mkdir newfolder
mkdir testfolder
cd newfolder
touch newfolder.txt newfolder1.txt
cd ../testfolder/
touch testfolder.txt testfolder1.txt
cd ..
touch sample4.java
ls

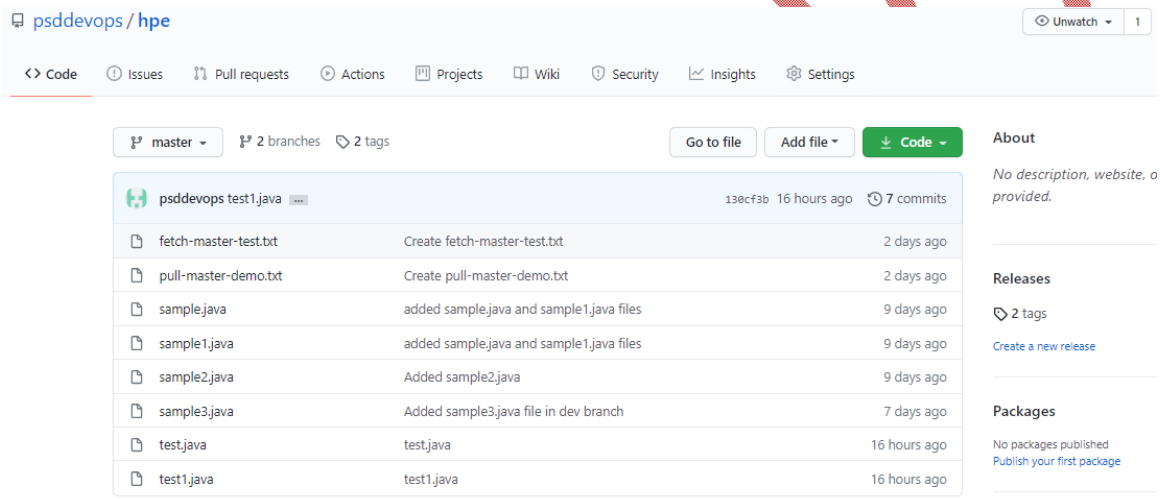
```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo
$ cd hpe/

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ touch sample.class test.class sample.conf test.conf .classpath .project

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ mkdir newfolder

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ mkdir testfolder

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ cd newfolder/

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe/newfolder (master)
$ touch newfolder.txt newfolder1.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe/newfolder (master)
$ cd ../testfolder/

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe/testfolder (master)
$ touch testfolder.txt testfolder1.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe/testfolder (master)
$ cd ..

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ ls
fetch-master-test.txt  pull-master-demo.txt  sample.conf  sample1.java  sample3.java  test.conf  test1.java
newfolder/             sample.class          sample.java  sample2.java  test.class    test.java  testfolder/

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ touch sample4.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$
```

Create .gitignore file and add below content

*.classpath
*.project
*.class
*.conf
/newfolder /*
/testfolder /*



Verify the remote repository



Push to the remote repository

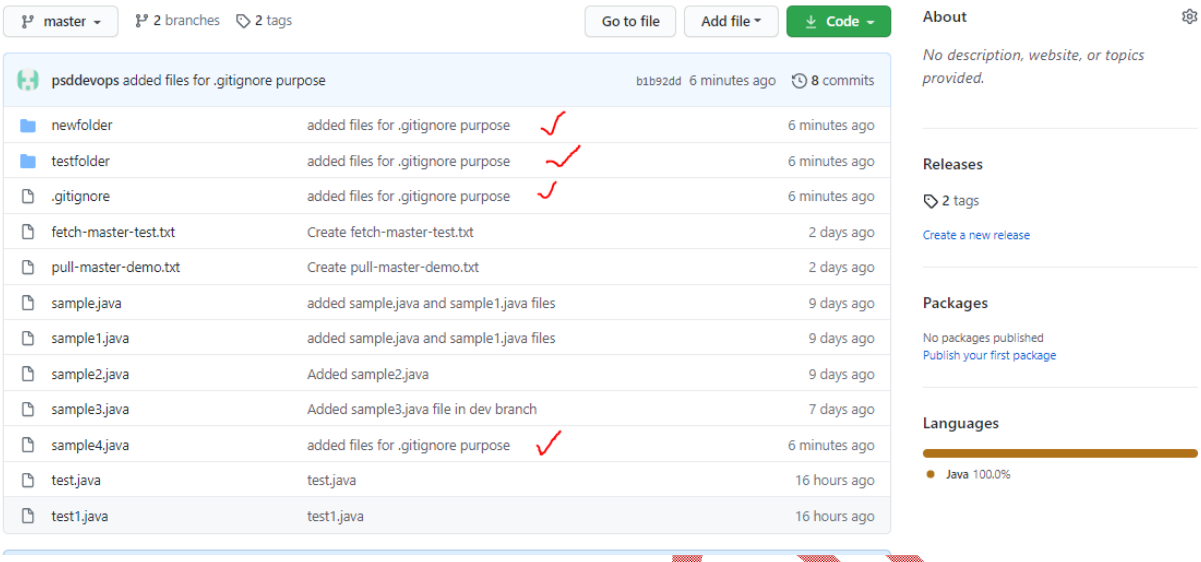
git status

git add . && git commit -m "added files for .gitignore purpose"

git push origin master

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (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)
        .gitignore
        newfolder/
        sample4.java
        testfolder/

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

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ ls
fetch-master-test.txt  pull-master-demo.txt  sample.conf  sample1.java  sample3.java  test.class  test.java   testfolder/
newfolder/             sample.class          sample.java  sample2.java  sample4.java  test.conf   test1.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git add . && git commit -m "added files for .gitignore purpose"
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory
[master b1b92dd] added files for .gitignore purpose
 6 files changed, 6 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 newfolder/newfolder.txt
 create mode 100644 newfolder/newfolder1.txt
 create mode 100644 sample4.java
 create mode 100644 testfolder/testfolder.txt
 create mode 100644 testfolder/testfolder1.txt
```

Verify the remote repo



Ignoring folder

mkdir samplefolder

cd samplefolder/

touch samplefolder.txt samplefolder1.txt samplefolder2.txt

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ mkdir samplefolder

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ cd samplefolder/

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe/samplefolder (master)
$ touch samplefolder.txt samplefolder1.txt samplefolder2.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe/samplefolder (master)
$ cd ..

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ ll
total 8
-rw-r--r-- 1 ppred 197609 23 Nov  7 10:07 fetch-master-test.txt
drwxr-xr-x 1 ppred 197609  0 Nov  7 10:14 newfolder/
-rw-r--r-- 1 ppred 197609 22 Nov  7 10:07 pull-master-demo.txt
-rw-r--r-- 1 ppred 197609  0 Nov  7 10:13 sample.class
-rw-r--r-- 1 ppred 197609  0 Nov  7 10:13 sample.conf
-rw-r--r-- 1 ppred 197609  0 Nov  7 10:07 sample.java
-rw-r--r-- 1 ppred 197609  0 Nov  7 10:07 sample1.java
-rw-r--r-- 1 ppred 197609  0 Nov  7 10:07 sample2.java
-rw-r--r-- 1 ppred 197609  0 Nov  7 10:07 sample3.java
-rw-r--r-- 1 ppred 197609  0 Nov  7 10:17 sample4.java
drwxr-xr-x 1 ppred 197609  0 Nov  7 10:43 samplefolder/
-rw-r--r-- 1 ppred 197609  0 Nov  7 10:13 test.class
-rw-r--r-- 1 ppred 197609  0 Nov  7 10:13 test.conf
-rw-r--r-- 1 ppred 197609 11 Nov  7 10:07 test.java
-rw-r--r-- 1 ppred 197609 12 Nov  7 10:07 test1.java
drwxr-xr-x 1 ppred 197609  0 Nov  7 10:16 testfolder/

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ vi .gitignore
```

vi .gitignore

```
*.classpath
*.project
*.class
*.conf
/newfolder /*
/testfolder /*
/samplefolder/*
~
~
```

git status

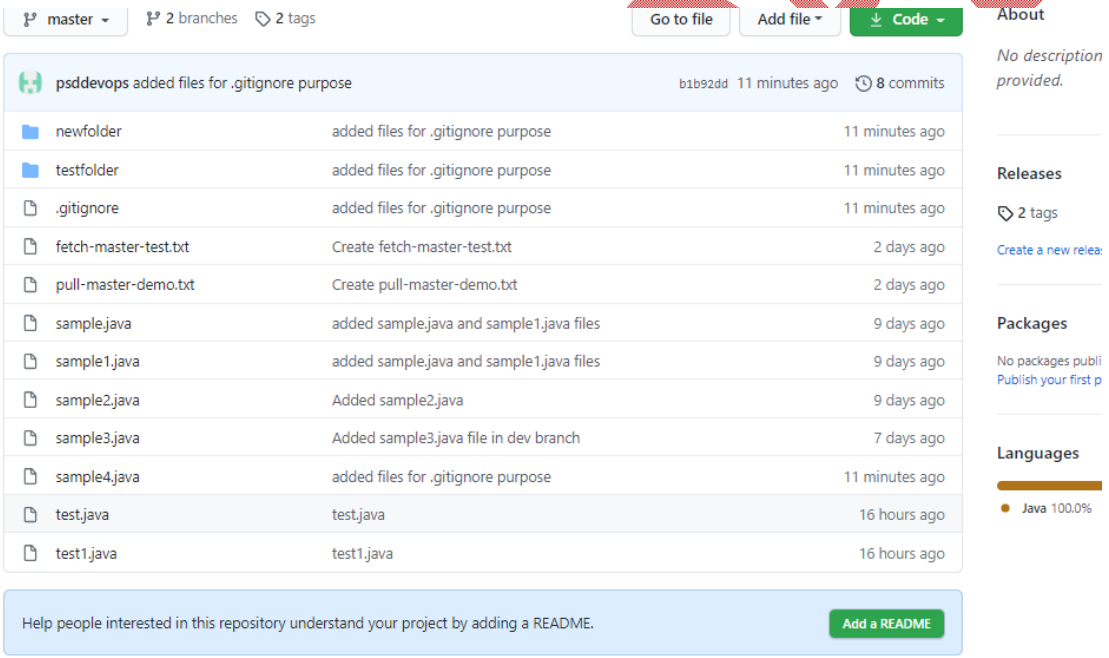
git add . && git commit -m "added sample folder"

git push origin master

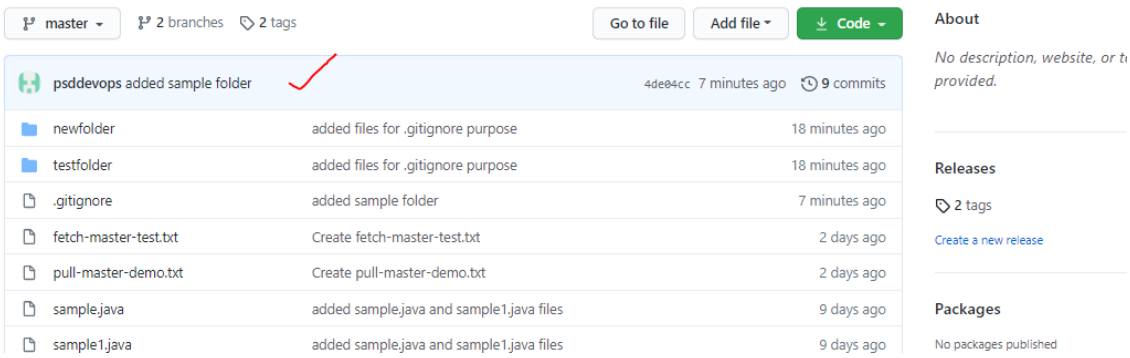
*Prepared by **PPNREDDY** .....*

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

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:   .gitignore

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

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git add . && git commit -m "added sample folder"
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory
[master 4de04cc] added sample folder
 1 file changed, 1 insertion(+)

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 322 bytes | 322.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/psddevops/hpe.git
   b1b92dd..4de04cc  master -> master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
```

Before pushing remote repo



After pushing remote repo



*Prepared by **PPNREDDY** .....*

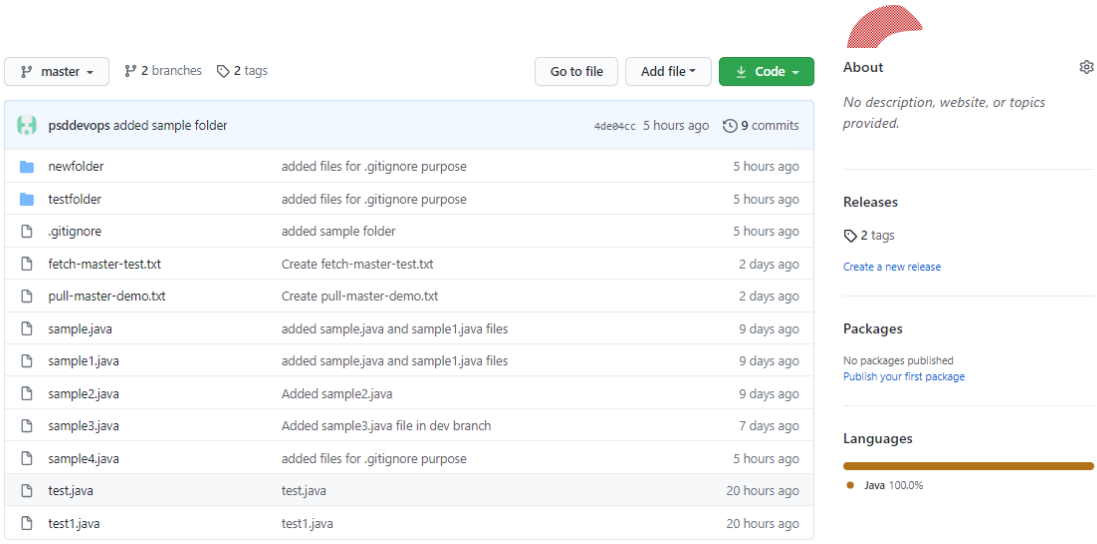
Removing a file/Directory from github/bitbucket repository

To delete file from git hub repository you must follow three steps.

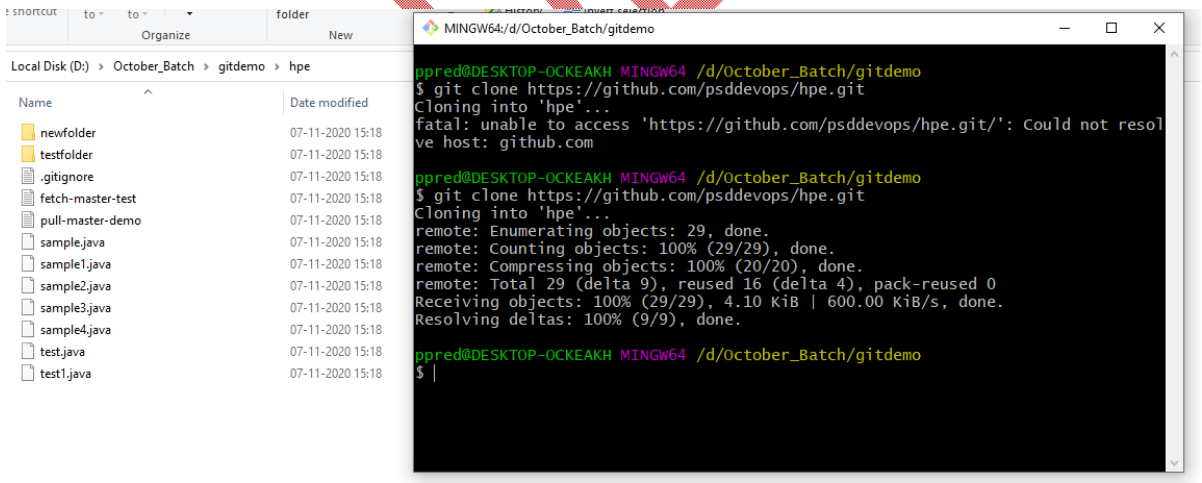
1. Remove file in local repository (working directory)
2. Commit your changes
3. Push your changes to remote git repository(github)

Removing files from remote repo

Files in remote repository



Clone the remote repository to local



ls

git rm fetch-master-test.txt pull-master-demo.txt

git status

git add . && git commit -m "deleted the files ...."

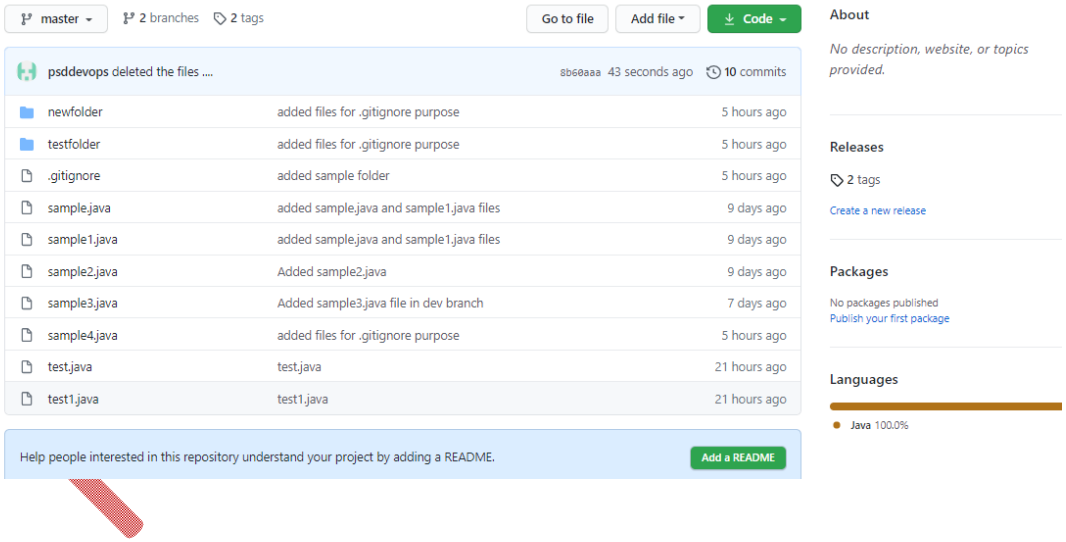
git push origin master

*Prepared by **PPNREDDY** .....*

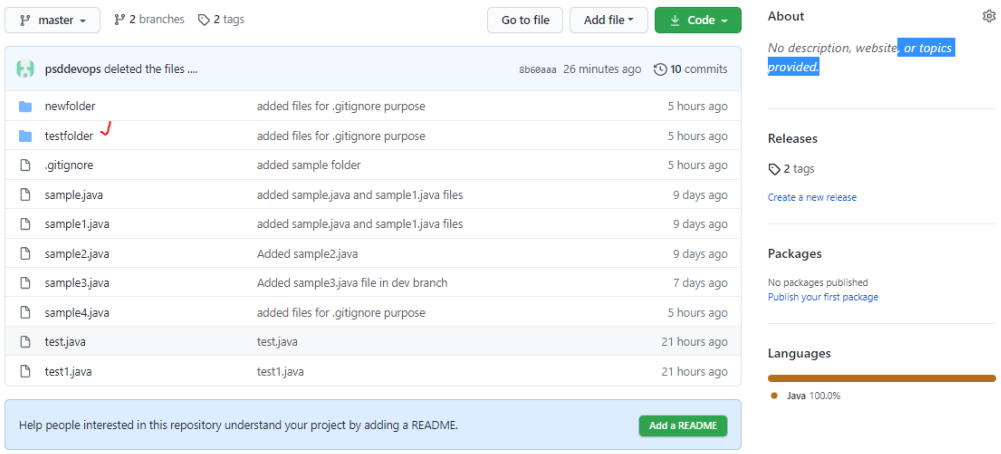
```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ ls
fetch-master-test.txt  pull-master-demo.txt  sample1.java  sample3.java  test.java   testfolder/
newfolder/             sample.java           sample2.java  sample4.java  test1.java

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git rm fetch-master-test.txt pull-master-demo.txt
rm 'fetch-master-test.txt'
rm 'pull-master-demo.txt'

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    fetch-master-test.txt
        deleted:    pull-master-demo.txt
```

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git add . && git commit -m "deleted the files ...."
[master 8b60aaa] deleted the files ....
 2 files changed, 2 deletions(-)
 delete mode 100644 fetch-master-test.txt
 delete mode 100644 pull-master-demo.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 236 bytes | 236.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/psddevops/hpe.git
   4de04cc..8b60aaa  master -> master

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/hpe (master)
$
```

Verify the remote repo/



Removing folder from remote repo



Removing *testfolder* in remote repo

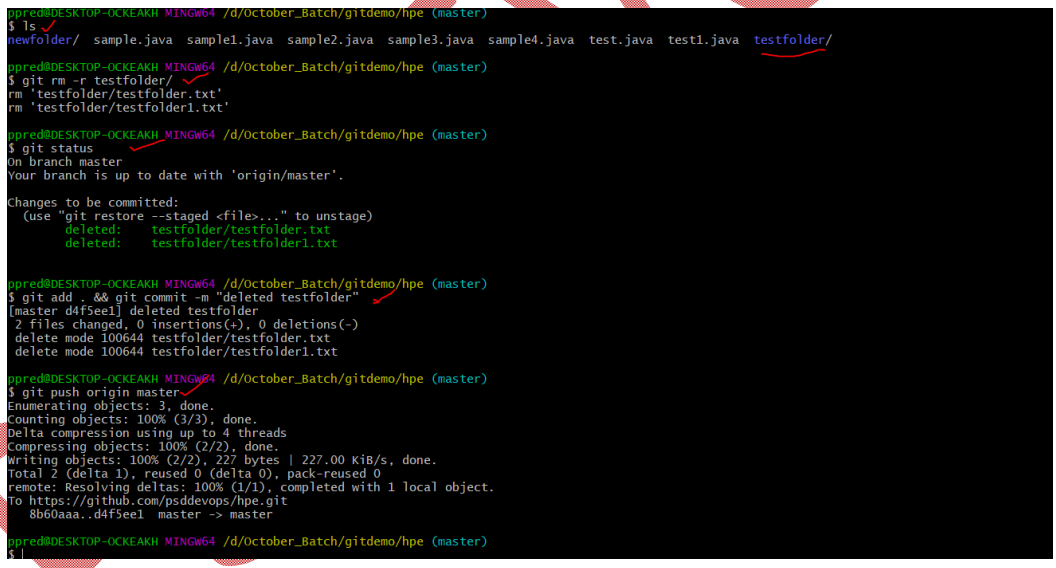
ls

git rm -r testfolder/

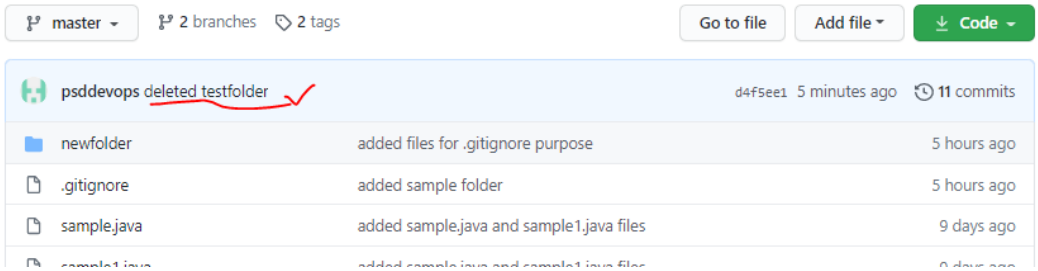
git status

git add . && git commit -m "deleted testfolder"

git push origin master



Verify the central repository



*Prepared by **PPNREDDY** .....*

Git Hooks/Triggers/Wrapper Scripts

Git Hooks are the scripts which trigger when you perform a specific action in Git. These are useful to automate the tasks.

Ex: When you create a Git Hook to run commit validation every time you commit code.

We can apply the script in two ways.

1. Before executing the command (Pre-step)
2. After executing the command (Post-step)

There are two types of git hooks.

1. Client-Side Hooks
2. Server-Side Hooks

Client-Side Hooks are trigged by operations such as committing and merging.

Server-Side Hooks are triggered by network operations such a receiving pushed commits.

After you are initialising the git repo, using git init command, it will create a one folder called. git and it contains some sub folders called hooks, branches, info, objects... etc

There are some predefined client-side hooks are available in hooks folder.

applypatch-msg.sample
post-update.sample
pre-push.sample
prepare-commit-msg.sample
commit-msg.sample
pre-applypatch.sample
pre-rebase.sample
update.sample
fsmonitor-watchman.sample
pre-commit.sample
pre-receive.sample

To enable any hook from above list, remove the ".sample" from each hook. Ex:

To enable pre-commit.sample hook, rename this file to "pre-commit".

https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks

```
# git init
# cd .git
# cd hooks
# ls –lrt
```

```
$ rm -rf .git

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo
$ git init
Initialized empty Git repository in D:/October_Batch/gitdemo/.git/

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ cd .git

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git (GIT_DIR!)
$ ls -lrt
total 7
-rw-r--r-- 1 ppred 197609  73 Nov  7 21:17 description
drwxr-xr-x 1 ppred 197609   0 Nov  7 21:17 hooks/
drwxr-xr-x 1 ppred 197609   0 Nov  7 21:17 info/
drwxr-xr-x 1 ppred 197609   0 Nov  7 21:17 refs/
-rw-r--r-- 1 ppred 197609  23 Nov  7 21:17 HEAD
drwxr-xr-x 1 ppred 197609   0 Nov  7 21:17 objects/
-rw-r--r-- 1 ppred 197609 130 Nov  7 21:17 config

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git (GIT_DIR!)
$ cd hooks

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$ ll
total 41
-rwxr-xr-x 1 ppred 197609  478 Nov  7 21:17 applypatch-msg.sample*
-rwxr-xr-x 1 ppred 197609  896 Nov  7 21:17 commit-msg.sample*
-rwxr-xr-x 1 ppred 197609 4655 Nov  7 21:17 fsmonitor-watchman.sample*
-rwxr-xr-x 1 ppred 197609  189 Nov  7 21:17 post-update.sample*
-rwxr-xr-x 1 ppred 197609  424 Nov  7 21:17 pre-applypatch.sample*
-rwxr-xr-x 1 ppred 197609 1643 Nov  7 21:17 pre-commit.sample*
-rwxr-xr-x 1 ppred 197609  416 Nov  7 21:17 pre-merge-commit.sample*
-rwxr-xr-x 1 ppred 197609 1492 Nov  7 21:17 prepare-commit-msg.sample*
-rwxr-xr-x 1 ppred 197609 1374 Nov  7 21:17 pre-push.sample*
-rwxr-xr-x 1 ppred 197609 4898 Nov  7 21:17 pre-rebase.sample*
-rwxr-xr-x 1 ppred 197609  544 Nov  7 21:17 pre-receive.sample*
-rwxr-xr-x 1 ppred 197609 3650 Nov  7 21:17 update.sample*

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$
```

Rename the sample extension files

mv pre-commit.sample pre-commit

mv commit-msg.sample commit-msg

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$ cp pre-commit.sample 1_pre-commit.sample

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$ mv pre-commit.sample pre-commit

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$ cp commit-msg.sample 1_commit-msg.sample

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$ mv commit-msg.sample commit-msg

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$
```

Update the *pre-commit* and *commit-msg* files

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$ vi commit-msg

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$ cat commit-msg
#!/bin/sh
#
# An example hook script to check the commit log message.
# Called by "git commit" with one argument, the name of the file
# that has the commit message.  The hook should exit with non-zero
# status after issuing an appropriate message if it wants to stop the
# commit.  The hook is allowed to edit the commit message file.
#
# To enable this hook, rename this file to "commit-msg".

# Uncomment the below to add a Signed-off-by line to the message.
# Doing this in a hook is a bad idea in general, but the prepare-commit-msg
# hook is more suited to it.
#
# SOB=$(git var GIT_AUTHOR_IDENT | sed -n 's/^\(.*>\).*$/Signed-off-by: \1/p')
# grep -qs "^$SOB" "$1" || echo "$SOB" >> "$1"

# This example catches duplicate Signed-off-by lines.

#test "" = "$(grep '^Signed-off-by: ' "$1" |
#        sort | uniq -c | sed -e '/^[   ]*1[    ]/d')" || {
#       echo >&2 Duplicate Signed-off-by lines.
#       exit 1
#}

echo "git commit successfuly completed ......................."

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
```

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$ vi pre-commit

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$ cat pre-commit
#!/bin/sh
#
# An example hook script to verify what is about to be committed.
# Called by "git commit" with no arguments.  The hook should
# exit with non-zero status after issuing an appropriate message if
# it wants to stop the commit.
#
# To enable this hook, rename this file to "pre-commit".
echo "This is pre-commit message ....."

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo/.git/hooks (GIT_DIR!)
$
```

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ ls
a.txt  b.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git status
On branch master

No commits yet

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

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

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git add .

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git commit -m " Added a.txt and b.txt files......."
This is pre-commit message .....
git commit successfuly completed .....
[master (root-commit) b591b23]  Added a.txt and b.txt files.......
 2 files changed, 2 insertions(+)
 create mode 100644 a.txt
 create mode 100644 b.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Checking the files modified between two commits

git show <commit-id> --name-only --pretty=""

git diff 68519a3b657 b591b23d2fc --name-only

```
ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ vi c.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git add . && git commit -m "added c.txt file"
warning: LF will be replaced by CRLF in c.txt.
The file will have its original line endings in your working directory
This is pre-commit message .....
git commit successfuly completed .....
[master 68519a3] added c.txt file
 1 file changed, 1 insertion(+)
 create mode 100644 c.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git log
commit 68519a3b6570cb9670745d15e02a9e78c430bf2a (HEAD -> master)
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Sat Nov 7 21:36:13 2020 +0530

    added c.txt file

commit b591b23d2fc52afabc7e1f97e26f1cbf0a0932d8
Author: Pushpanath <ppreddy2711@gmail.com>
Date:   Sat Nov 7 21:32:48 2020 +0530

     Added a.txt and b.txt files.......

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$ git diff 68519a3b657 b591b23d2fc --name-only
c.txt

ppred@DESKTOP-OCKEAKH MINGW64 /d/October_Batch/gitdemo (master)
$
```

Git GUI

Git GUI is Tcl/Tk based graphical user interface to Git. It focuses on allowing users to make changes to their repository by making new commits, amending existing ones, creating branches, performing local merges, and fetching/pushing to remote repositories.

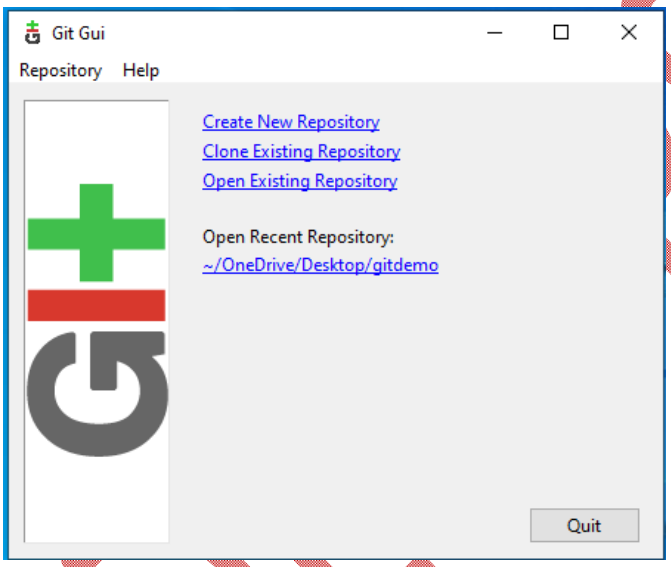
The first thing we need to do is install Git on Windows; you can do so with the following steps:

Step 1: Download and install the latest version of Git for Windows.

Step 2: Use the default options for each step in the installation.

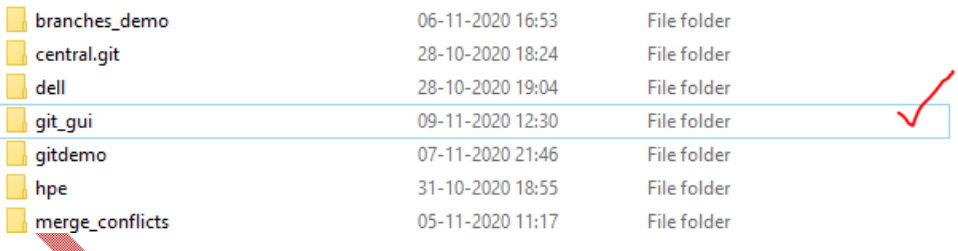
Step 3: Remove Git Bash Desktop Icon.

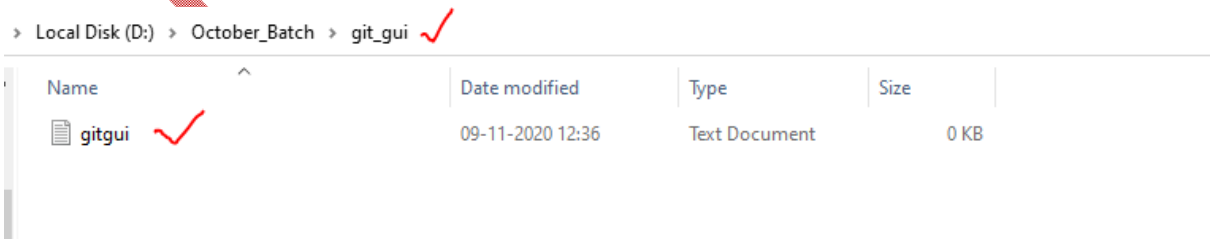
Step 4: Go to Start > All Programs > Git > Git GUI and make a Desktop Shortcut.

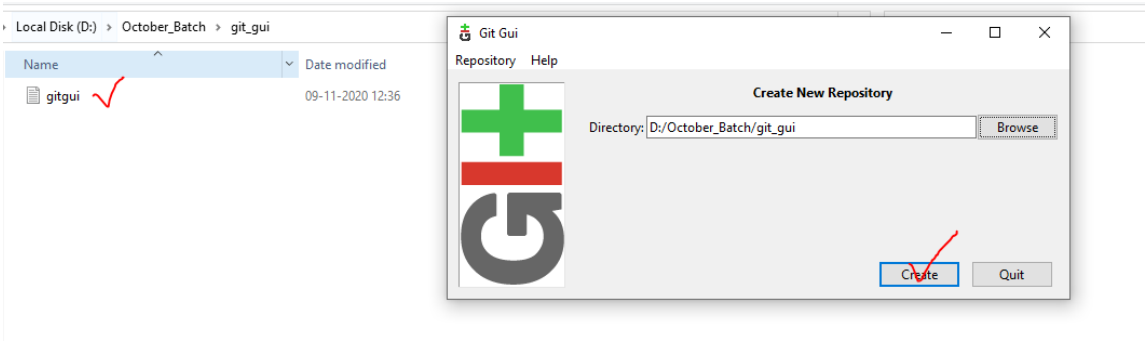


Create new repository in local

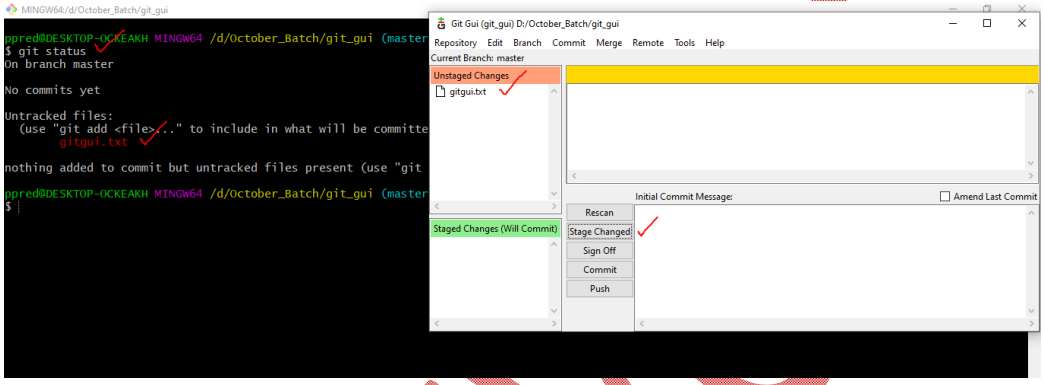
Create a folder *git-gui* and file in local



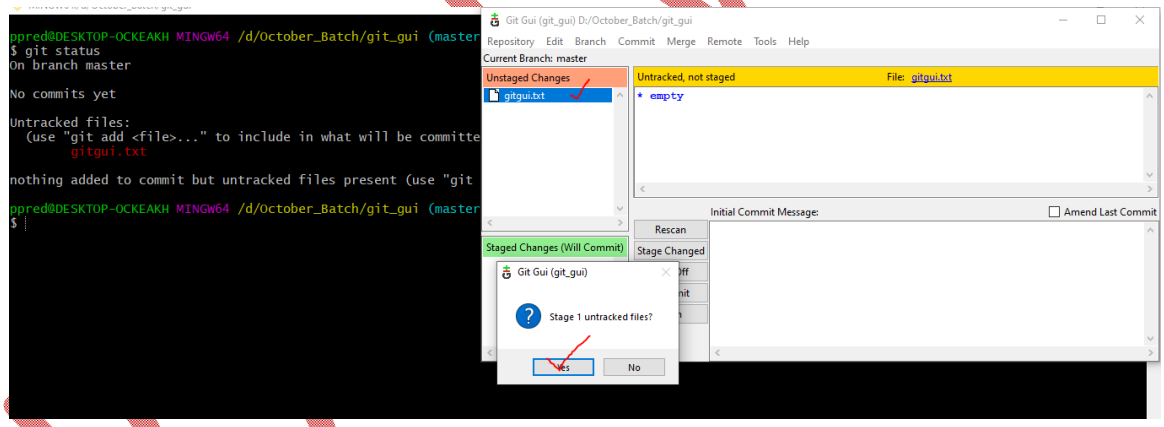


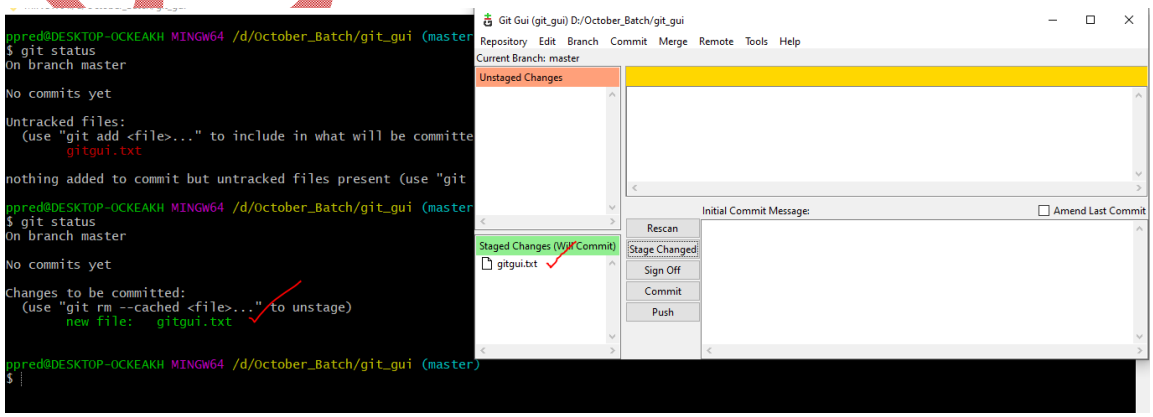


Click on the *file* and *Stage Changed*



Click on *Yes* it's equal to *# git add .*

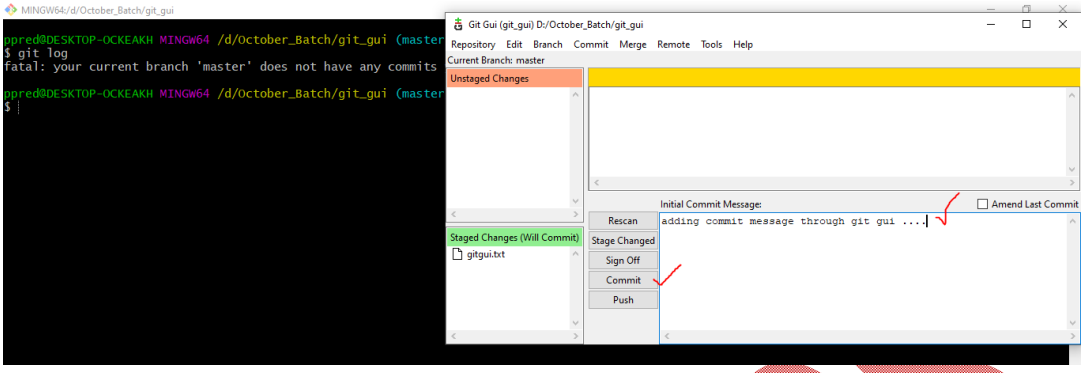




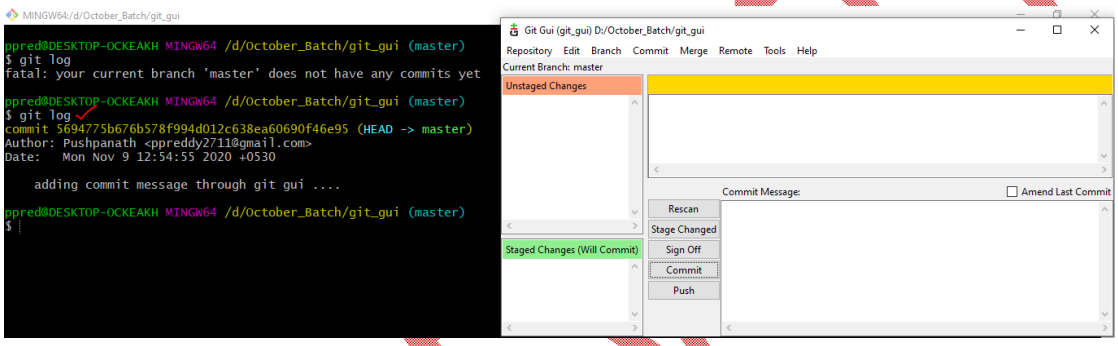
*Prepared by **PPNREDDY** .....*

Enter *commit message* and click *commit* button.

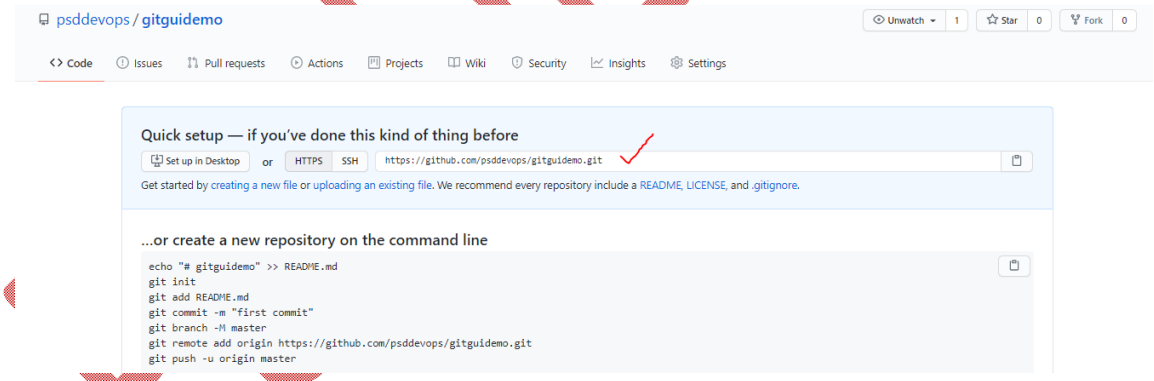
It's equal to #git commit –m "adding commit message through git gui ...."



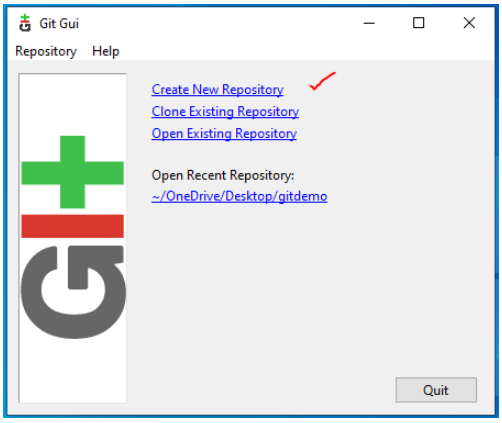
Verify the *logs*



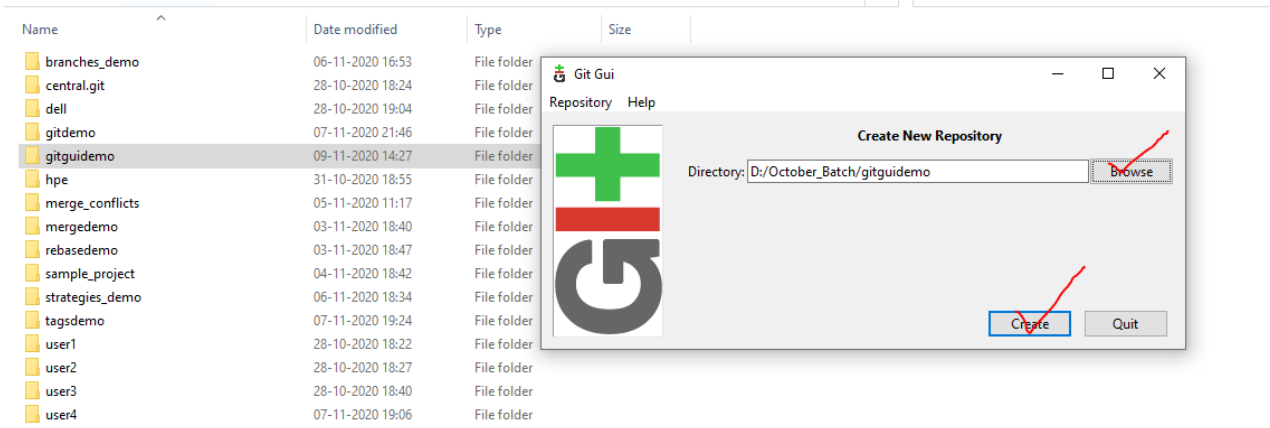
Create a remote repository in *github (https://github.com/psddevops/gitguidemo.git)*



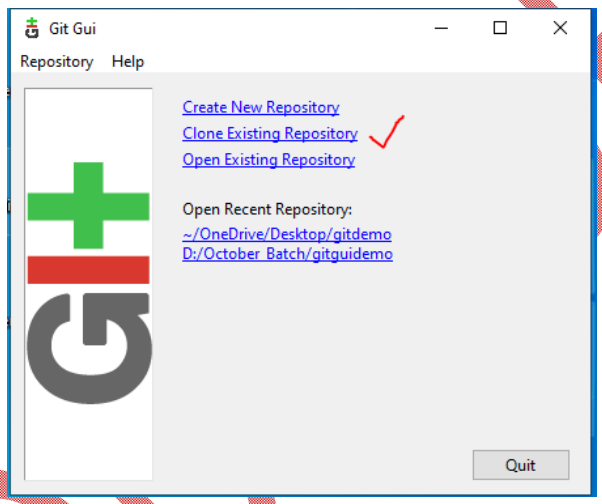
Create New repository



*Prepared by **PPNREDDY** .....*



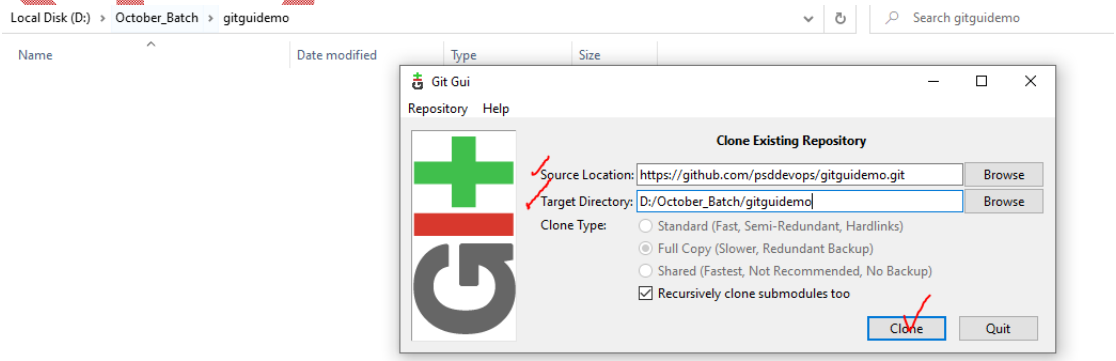
Clone a Remote Repository to a Local Repository



Enter Source Location and Target Directory

Source Location= https://github.com/psddevops/gitguidemo.git

Target_Directory= D:/October_Batch/gitguidemo



Exchanging the keys from server to github/Bitbucket

SSH keys are a way to identify trusted computers without involving passwords. You can generate an SSH key and add the public key to your GitHub account.

Generate ssh key using below command.

ssh-keygen

```
ubuntu@ip-172-31-3-62:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_rsa
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:KjG7NCVu5h0xFg78HOYuIb5lXcKvey8J8r5kQMOkTJo ubuntu@ip-172-31-3-62
The key's randomart image is:
+---[RSA 3072]----+
|   . .           |
| = =             |
|E o * +          |
|   . X o         |
|  . * % S        |
| . o.%.B         |
|  . %oB...       |
|   0 0.o+        |
|  . o.*+ o.      |
+----[SHA256]-----+
ubuntu@ip-172-31-3-62:~$
```

Verify the keys generated path

/home/ubuntu/.ssh

```
ubuntu@ip-172-31-3-62:~/.ssh$ pwd
/home/ubuntu/.ssh
ubuntu@ip-172-31-3-62:~/.ssh$ ls -lrt
total 12
-rw------- 1 ubuntu ubuntu  391 Nov  9 06:35 authorized_keys
-rw-r--r-- 1 ubuntu ubuntu  575 Nov  9 15:28 id_rsa.pub
-rw------- 1 ubuntu ubuntu 2610 Nov  9 15:28 id_rsa
ubuntu@ip-172-31-3-62:~/.ssh$
```

Verify and configure the git

vi /home/ubuntu/.gitconfig

git config --global –list

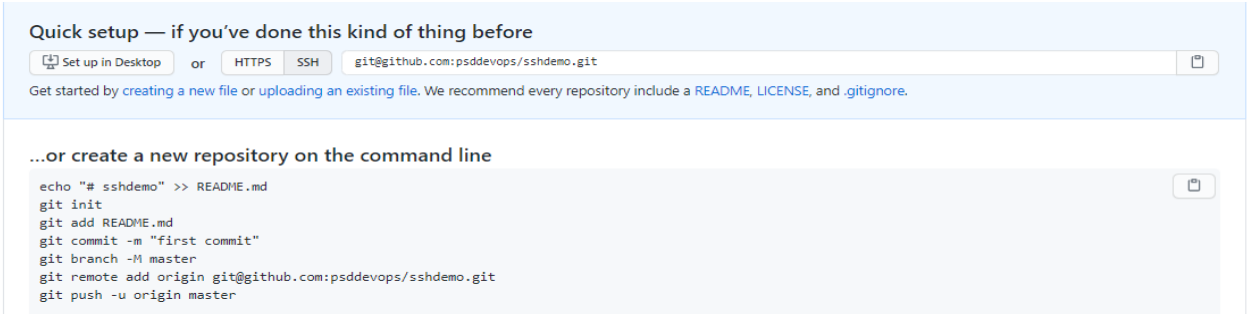
git config --global user.name "Pushpanath"

git config --global user.email “ppreddy2711@gmail.com”

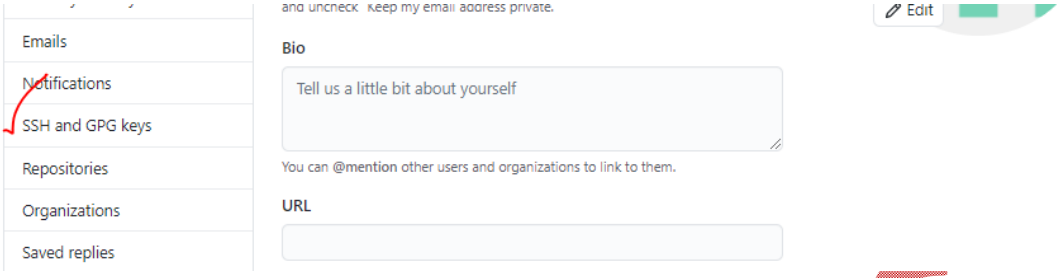
git config --global –list

```
ubuntu@ip-172-31-3-62:~/.ssh$
ubuntu@ip-172-31-3-62:~/.ssh$ git config --global --list
fatal: unable to read config file '/home/ubuntu/.gitconfig': No such file or directory
ubuntu@ip-172-31-3-62:~/.ssh$ vi /home/ubuntu/.gitconfig
ubuntu@ip-172-31-3-62:~/.ssh$ git config --global --list
ubuntu@ip-172-31-3-62:~/.ssh$ git config --global user.name "Pushpanath"
ubuntu@ip-172-31-3-62:~/.ssh$ git config --global user.email "ppreddy2711@gmail.com"
ubuntu@ip-172-31-3-62:~/.ssh$ git config --global --list
user.name=Pushpanath
user.email=ppreddy2711@gmail.com
ubuntu@ip-172-31-3-62:~/.ssh$
```

Create repo in github and exchange the keys

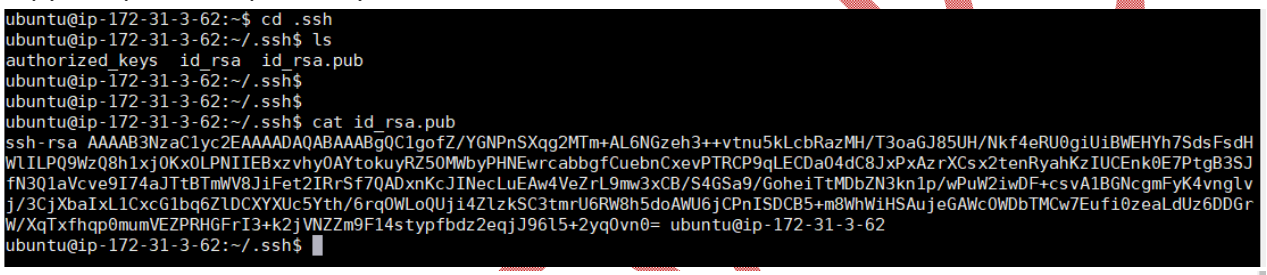


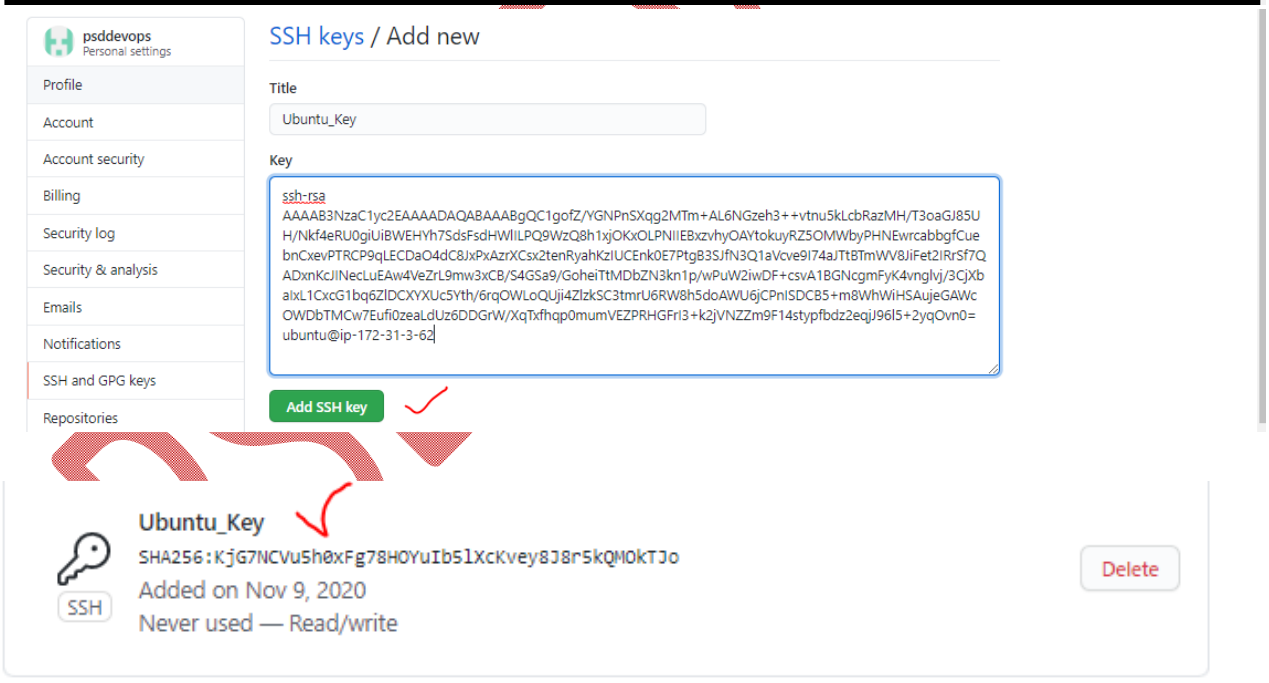
Click on  → Settings → SSH and GPG keys



Click on New ssh key

Copy and paste the publickey





Verify the sample application

mkdir gitdemo

cd gitdemo

git init

touch a.txt

git status

git add . && git commit -m "added a.txt file"

*Prepared by **PPNREDDY** .....*

```
ubuntu@ip-172-31-3-62:~$ mkdir gitdemo
ubuntu@ip-172-31-3-62:~$ cd gitdemo
ubuntu@ip-172-31-3-62:~/gitdemo$ git init
Initialized empty Git repository in /home/ubuntu/gitdemo/.git/
ubuntu@ip-172-31-3-62:~/gitdemo$ touch a.txt
ubuntu@ip-172-31-3-62:~/gitdemo$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
ubuntu@ip-172-31-3-62:~/gitdemo$ git add . && git commit -m "added a.txt file"
[master (root-commit) 04a455a] added a.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt
ubuntu@ip-172-31-3-62:~/gitdemo$
```

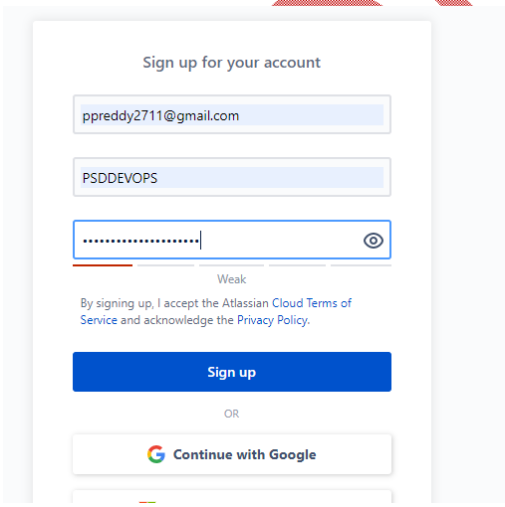
git remote add origin git@github.com:psddevops/sshdemo.git

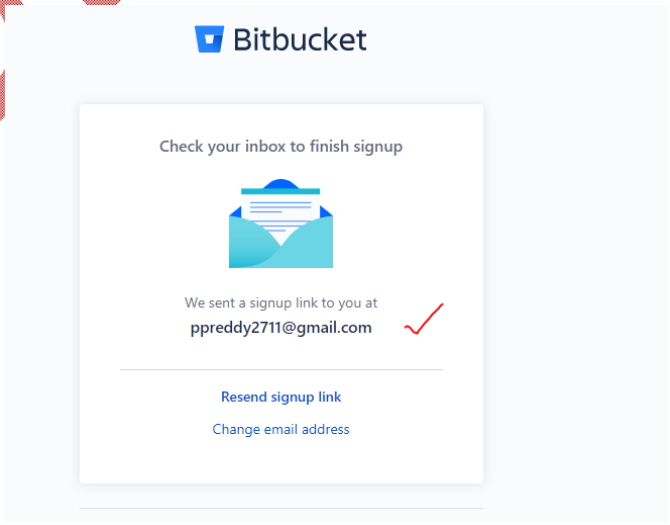
git push origin master

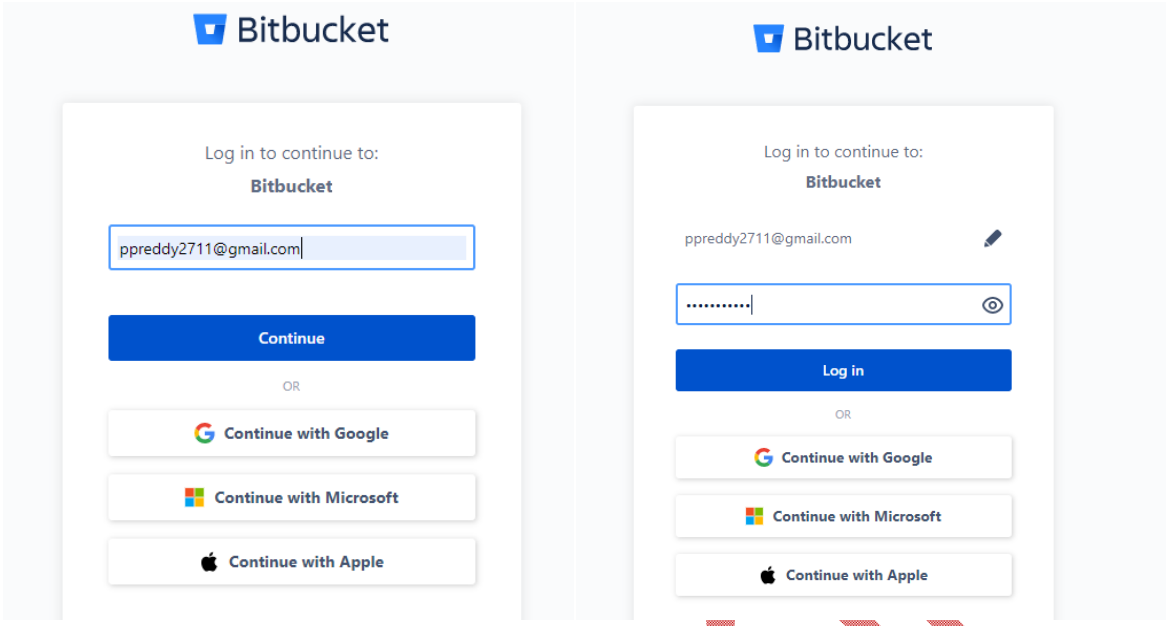
```
ubuntu@ip-172-31-3-62:~/gitdemo$ git remote add origin git@github.com:psddevops/sshdemo.git
ubuntu@ip-172-31-3-62:~/gitdemo$ git remote -v
origin  git@github.com:psddevops/sshdemo.git (fetch)
origin  git@github.com:psddevops/sshdemo.git (push)
ubuntu@ip-172-31-3-62:~/gitdemo$ git push origin master
Warning: Permanently added the RSA host key for IP address '140.82.113.3' to the list of known hosts.
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 215 bytes | 215.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:psddevops/sshdemo.git
 * [new branch]      master -> master
ubuntu@ip-172-31-3-62:~/gitdemo$
```

Integrate with Bitbucket from GIT

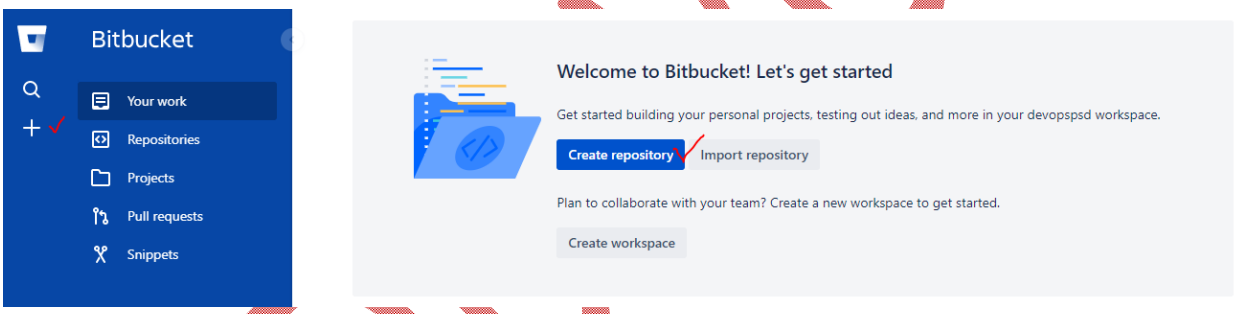
Sign in to the Bitbucket

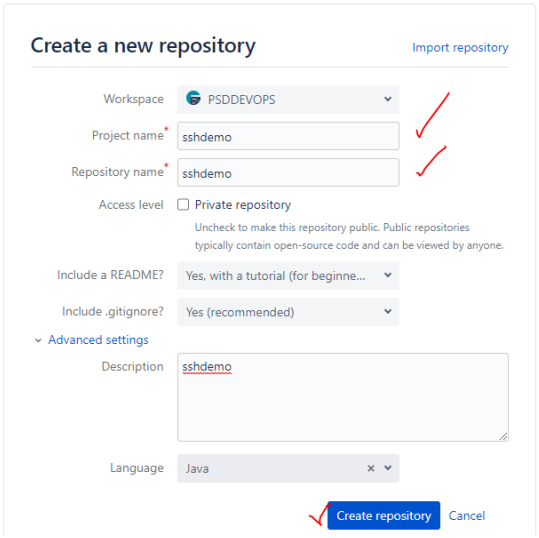


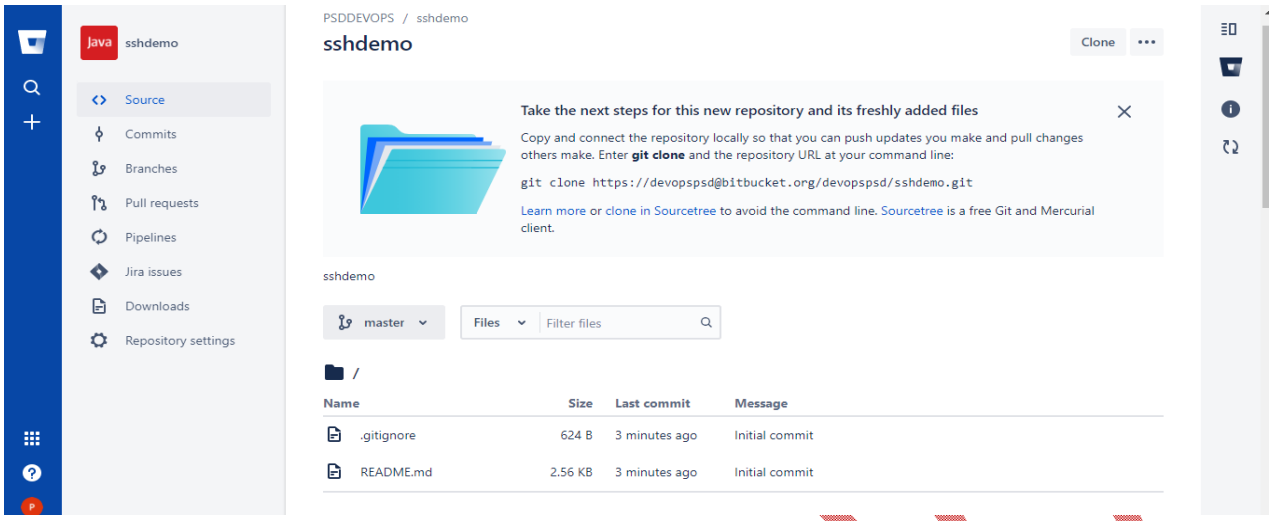


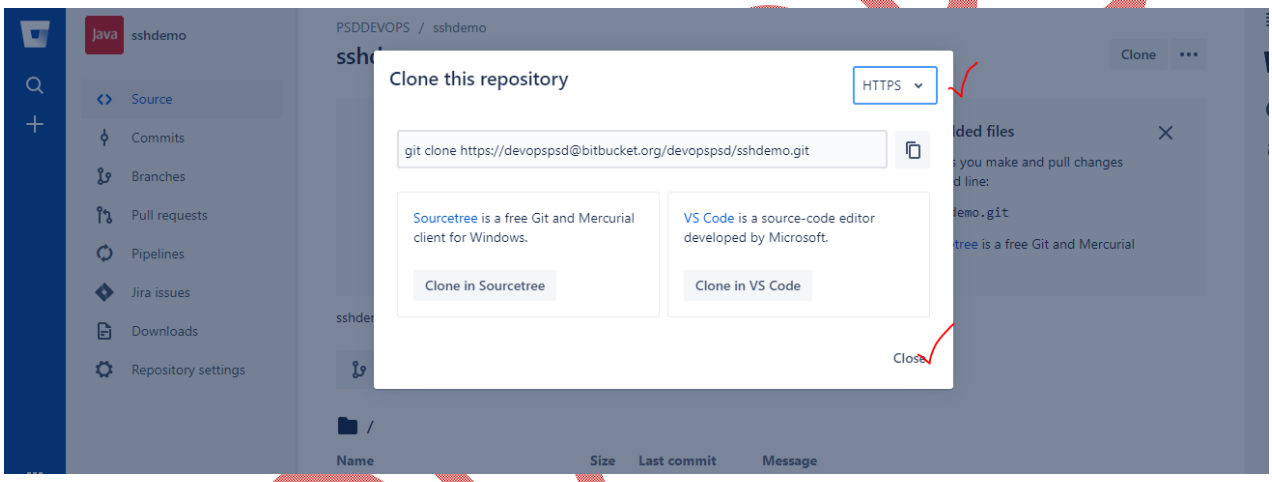


Create repo in Bitbucket and exchange the keys









Exchanging the keys between server and Bitbucket

cd /home/ubuntu/.ssh

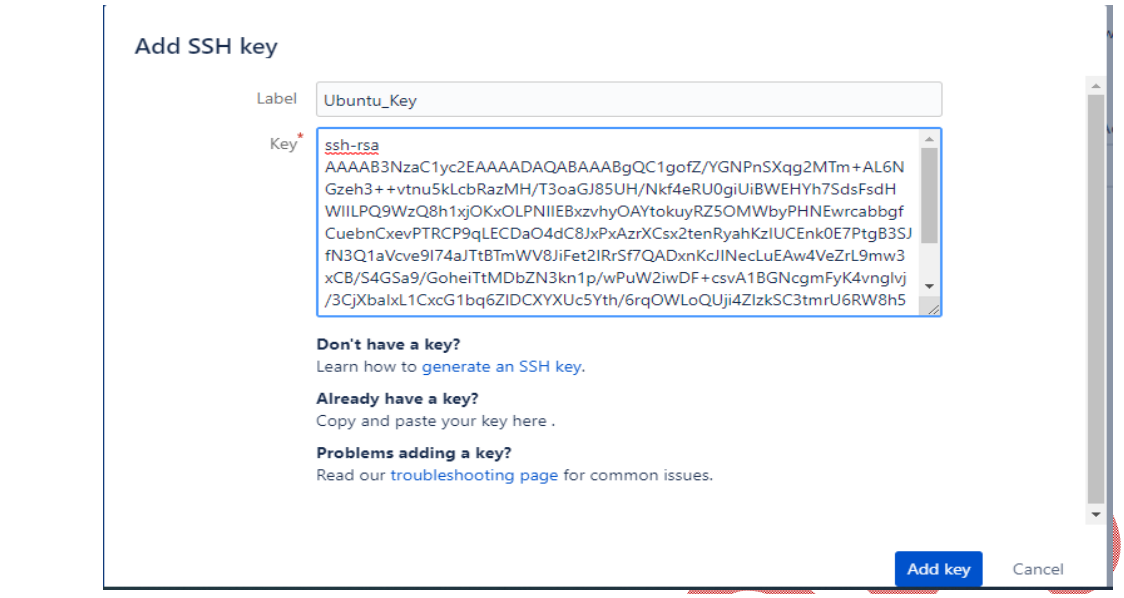
cat id_rsa.pub

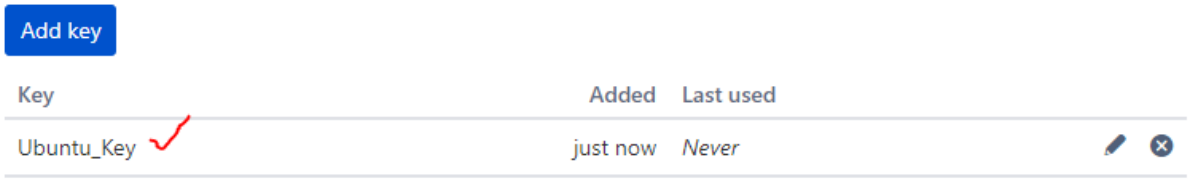
```
ubuntu@ip-172-31-3-62:~$ cd .ssh
ubuntu@ip-172-31-3-62:~/.ssh$ ls
authorized_keys  id_rsa  id_rsa.pub  known_hosts
ubuntu@ip-172-31-3-62:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC1gofZ/YGNPnSXqg2MTm+AL6NGzeh3++vtnu5kLcbRazMH/T3oaGJ85UH/Nkf4eRU0giUiBWEHYh7SdsFsdH
WlILPQ9WzQ8h1xjOKxOLPNIIEBxzvhyOAYtokuyRZ50MWbyPHNEwrcabbgfCuebnCxevPTRCP9qLECDaO4dC8JxPxAzrXCsx2tenRyahKzIUCEnk0E7PtgB3SJ
fN3Q1aVcve9I74aJTtBTmWV8JiFet2IRrSf7QADxnKcJINecLuEAw4VeZrL9mw3xCB/S4GSa9/GoheiTtMDbZN3kn1p/wPuW2iwDF+csvA1BGNcgmFyK4vnglv
j/3CjXbaIxL1CxcG1bq6ZlDCXYXUc5Yth/6rqOWLoQUji4ZlzkSC3tmrU6RW8h5doAWU6jCPnISDCB5+m8WhWiHSAujeGAWcOWDbTMCw7Eufi0zeaLdUz6DDGr
W/XqTxfhqp0mumVEZPRHGFrI3+k2jVNZZm9F14stypfbdz2eqjJ96l5+2yqOvn0= ubuntu@ip-172-31-3-62
ubuntu@ip-172-31-3-62:~/.ssh$
```

Adding ssh key to the Bitbucket

Click on →  icon → Personal Settings → SSH Keys → Add key

*Prepared by **PPNREDDY** .....*





git clone git@bitbucket.org:devopspsd/sshdemo.git
cd sshdemo
touch a.txt
git add . && git commit -m "added a.txt file"
git push origin master

```
ubuntu@ip-172-31-3-62:~/bitbucketdemo$ git clone git@bitbucket.org:devopspsd/sshdemo.git
Cloning into 'sshdemo'...
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (4/4), done.
ubuntu@ip-172-31-3-62:~/bitbucketdemo$ ll
total 12
drwxrwxr-x 3 ubuntu ubuntu 4096 Nov 10 08:10 ./
drwxr-xr-x 6 ubuntu ubuntu 4096 Nov 10 07:52 ../
drwxrwxr-x 3 ubuntu ubuntu 4096 Nov 10 08:10 sshdemo/
ubuntu@ip-172-31-3-62:~/bitbucketdemo$ cd sshdemo/
ubuntu@ip-172-31-3-62:~/bitbucketdemo/sshdemo$ ls
README.md
ubuntu@ip-172-31-3-62:~/bitbucketdemo/sshdemo$ touch a.txt
ubuntu@ip-172-31-3-62:~/bitbucketdemo/sshdemo$ git add . && git commit -m "added a.txt file"
[master 2cf326d] added a.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt
```

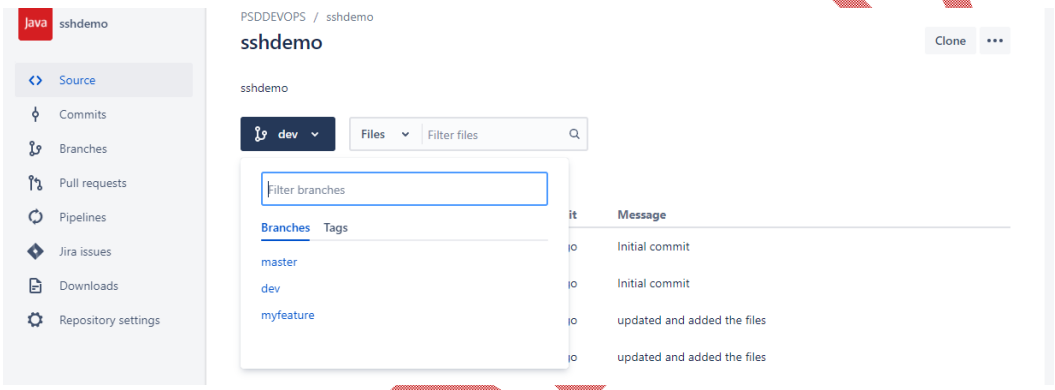
```
ubuntu@ip-172-31-3-62:~/bitbucketdemo/sshdemo$ git remote -v
origin  git@bitbucket.org:devopspsd/sshdemo.git (fetch)
origin  git@bitbucket.org:devopspsd/sshdemo.git (push)
ubuntu@ip-172-31-3-62:~/bitbucketdemo/sshdemo$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 311 bytes | 311.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To bitbucket.org:devopspsd/sshdemo.git
   21dd546..2cf326d  master -> master
ubuntu@ip-172-31-3-62:~/bitbucketdemo/sshdemo$
```

Pull Request

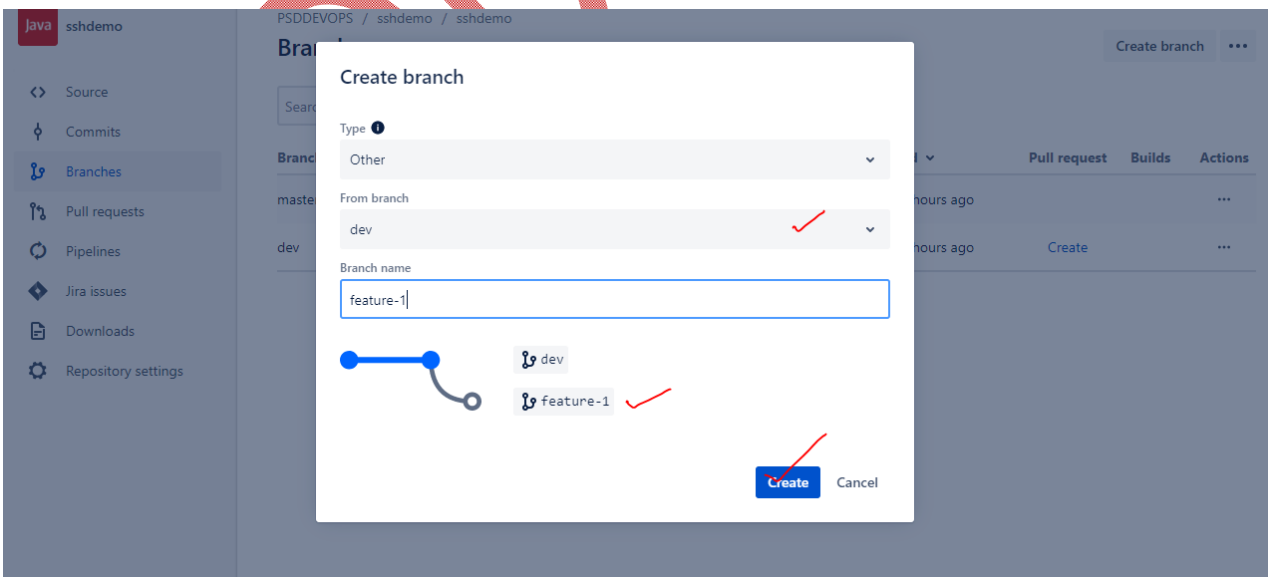
Pull request really a just a request that someone else review the work that you have done and merge our changes. Pull request enables team mates to review and comment on the changes before merging to main branch, we also can see how many file are modified, we also can compare modified file with their old version. When you create pull request, you have select two branches on GitHub/Bitbucket, The branch that you have made the changes on and where you would want to merge your changes into. Because Pull request occurs in GitHub/Bitbucket you would need to push your branch GitHub/Bitbucket before you create the request.

PR-Demo

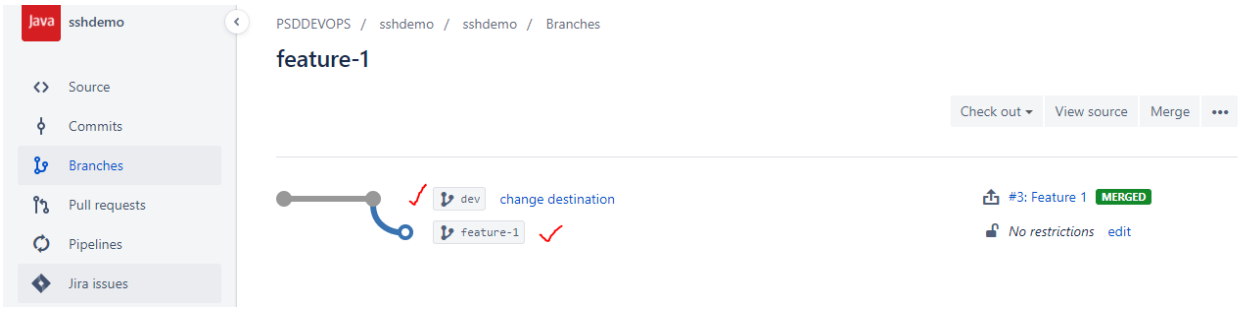
Create a feature branch from "dev" branch



Go to Branches → Create branch →Name: *feature-1* → Create



*Prepared by **PPNREDDY** .....*



Clone the *feature-1* branch and update the required changes

git clone -b feature-1 https://devopspsd@bitbucket.org/devopspsd/sshdemo.git

cd sshdemo/

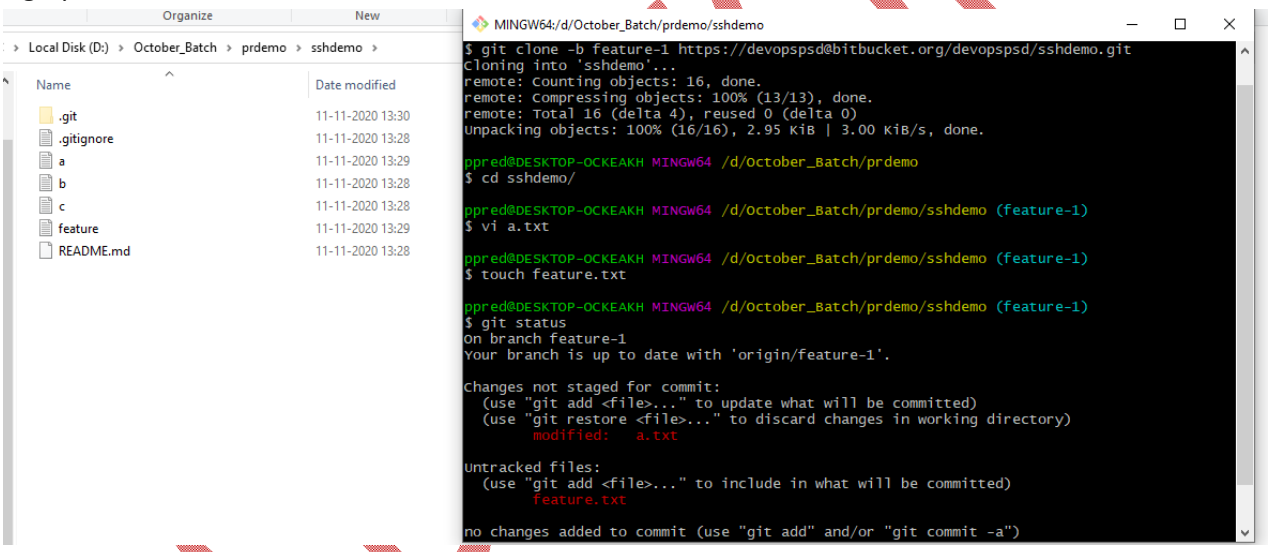
vi a.txt

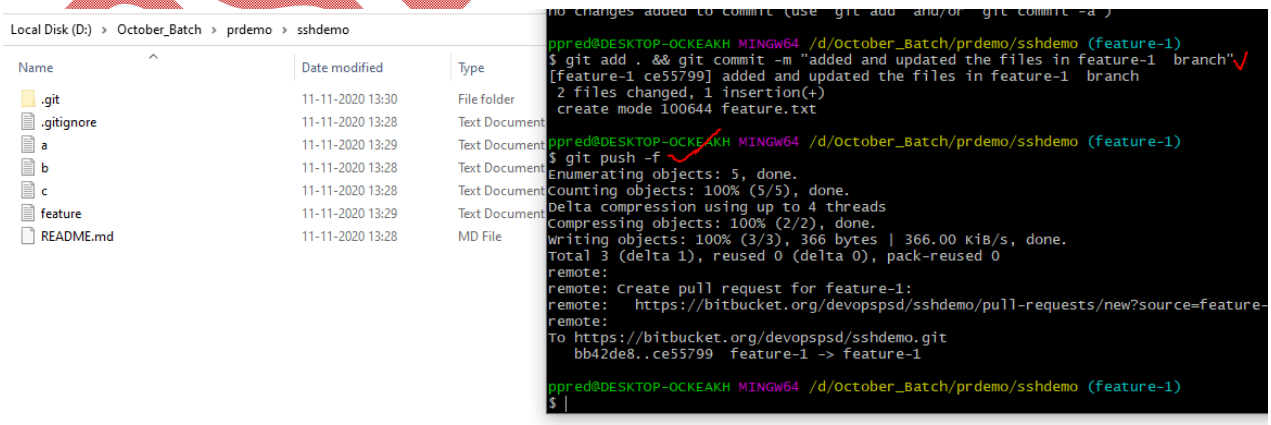
touch feature.txt

git status

git add . && git commit -m "added and updated the files in feature-1 branch"

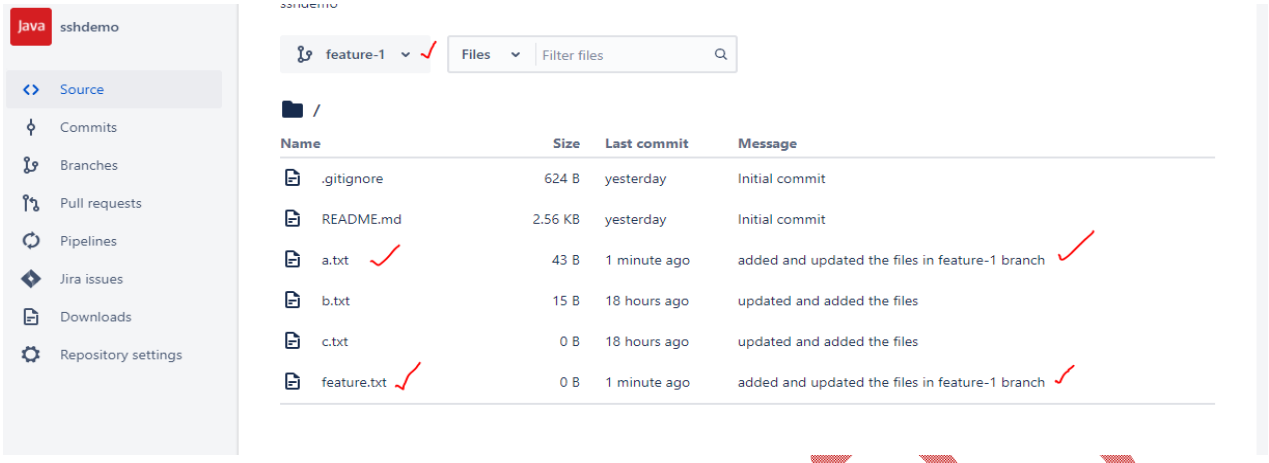
git push –f





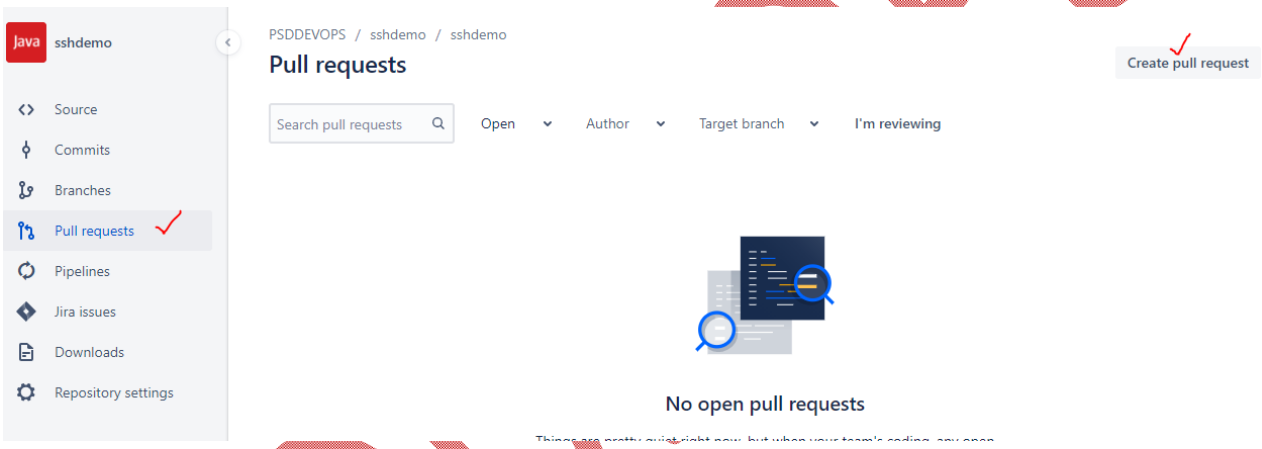
*Prepared by **PPNREDDY** .....*

Verify the remote *feature-1* branch in Bitbucket

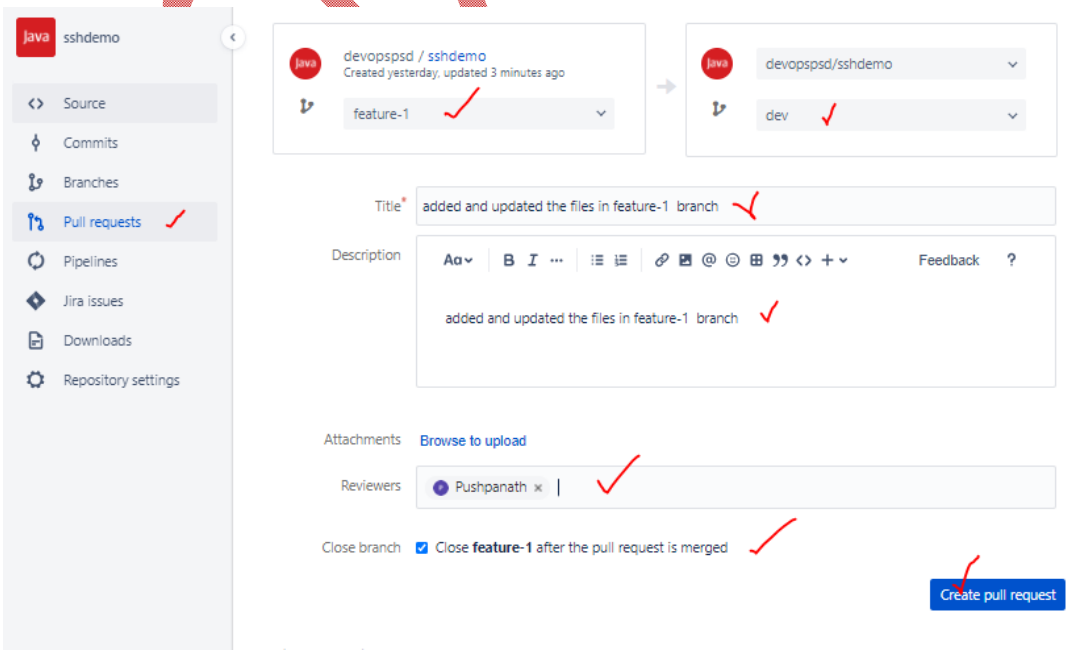


Create the *Pull Request* in Bitbucket

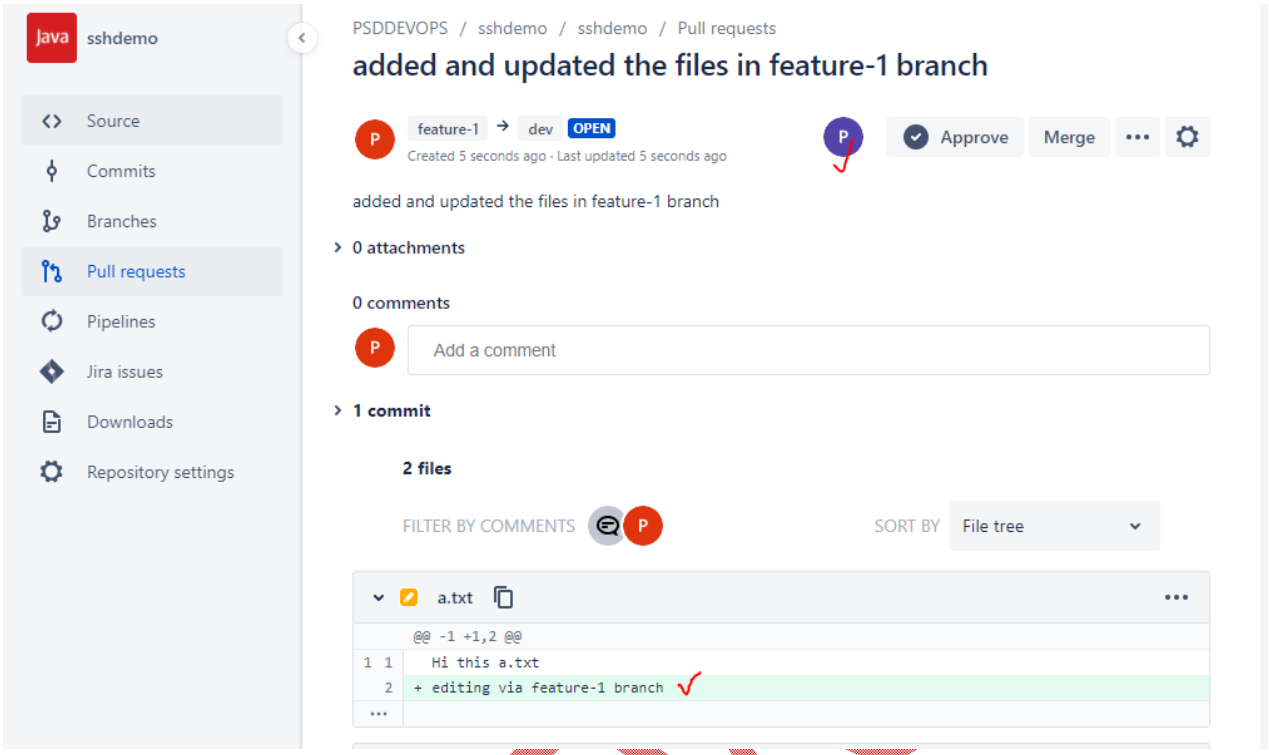
Click on Pull Requests → Create Pull Request



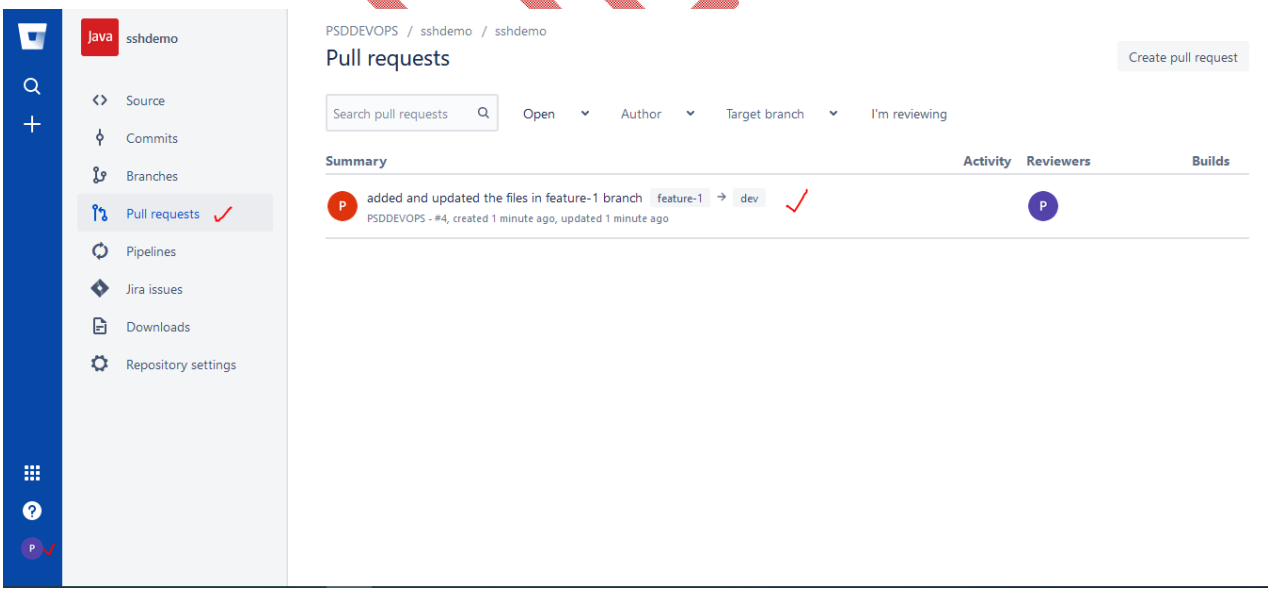
Adding the target branch (dev) reviewers (Pushpanath) like below → Create Pull Request

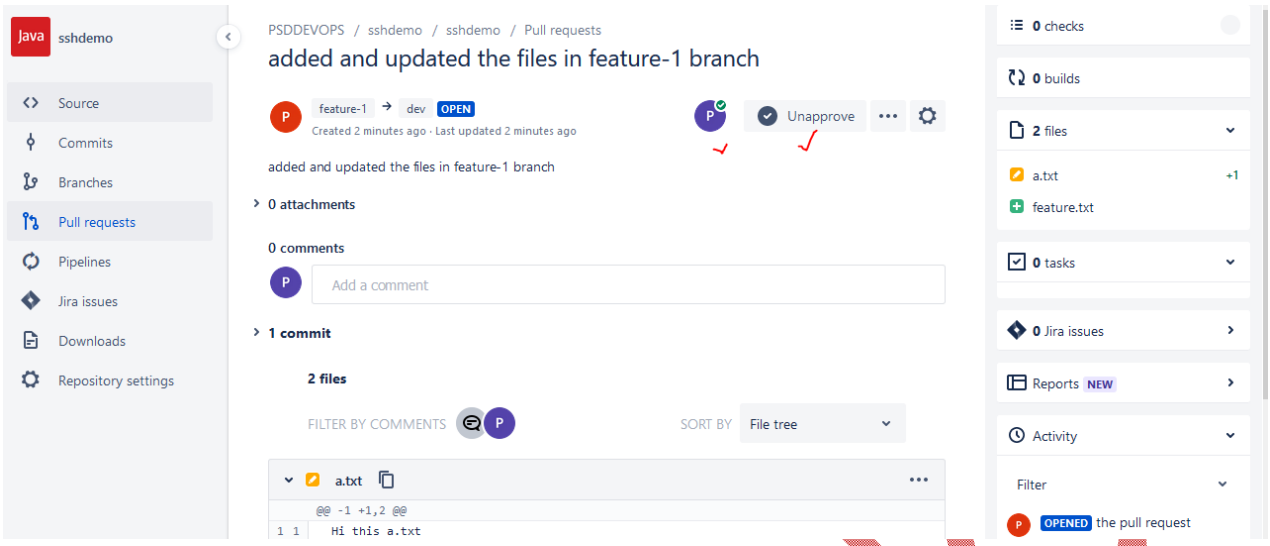


*Prepared by **PPNREDDY** .....*



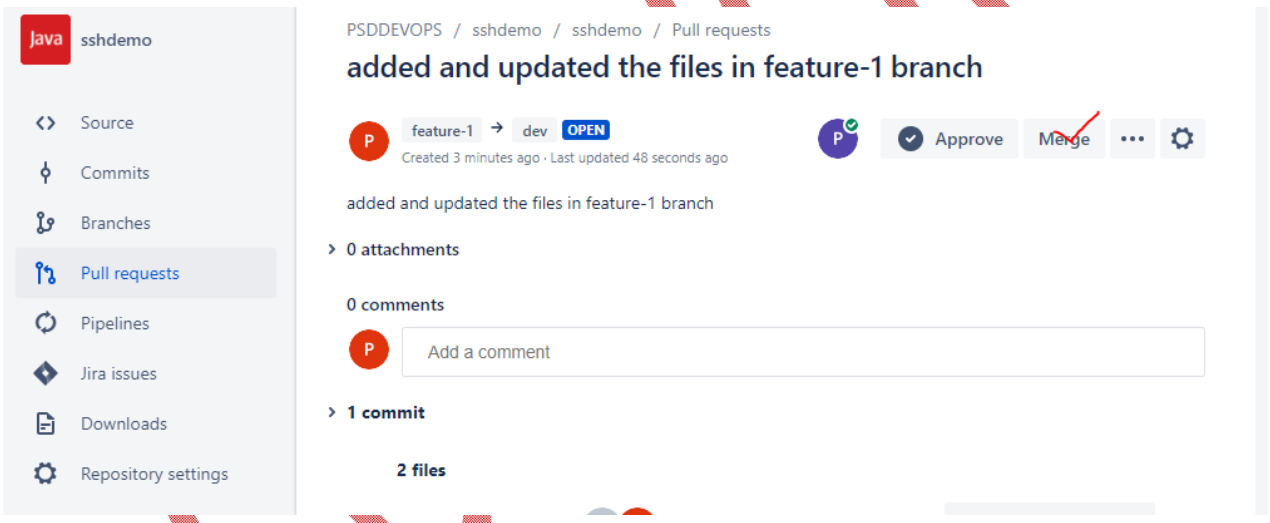
Reviewer will review and approve the PR

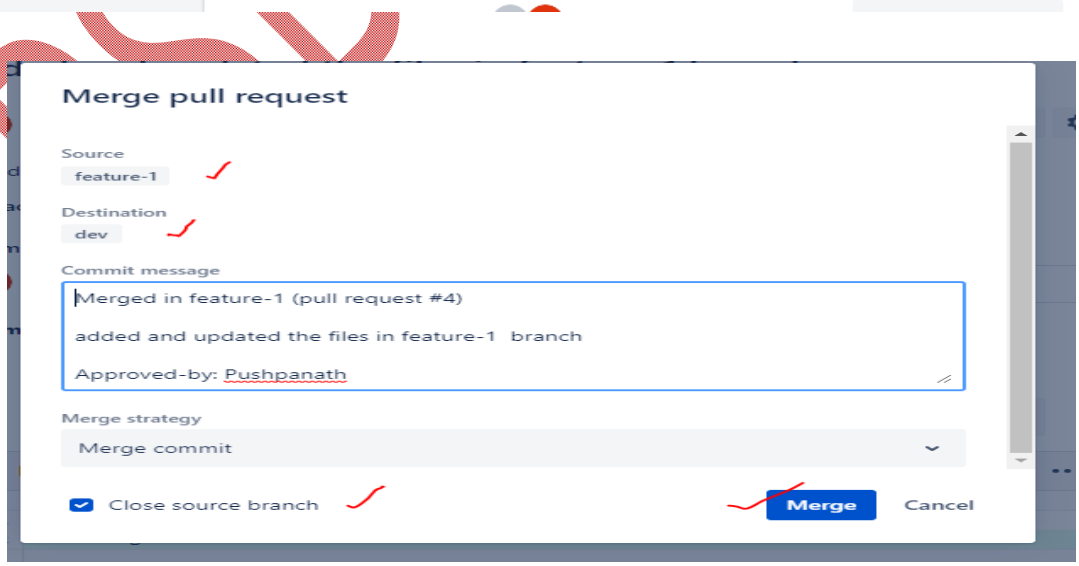




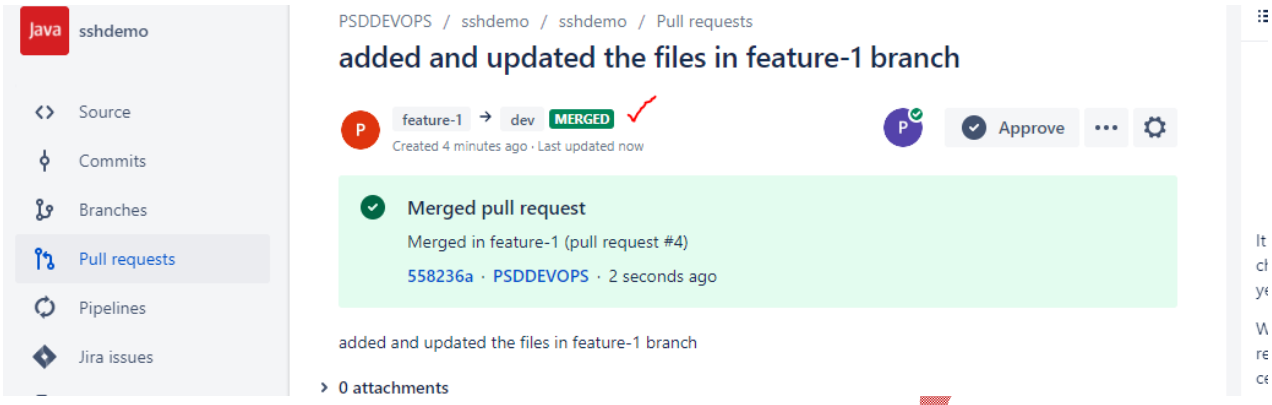
Merge the *feature-1* branch to *dev* branch.

Click on merge button

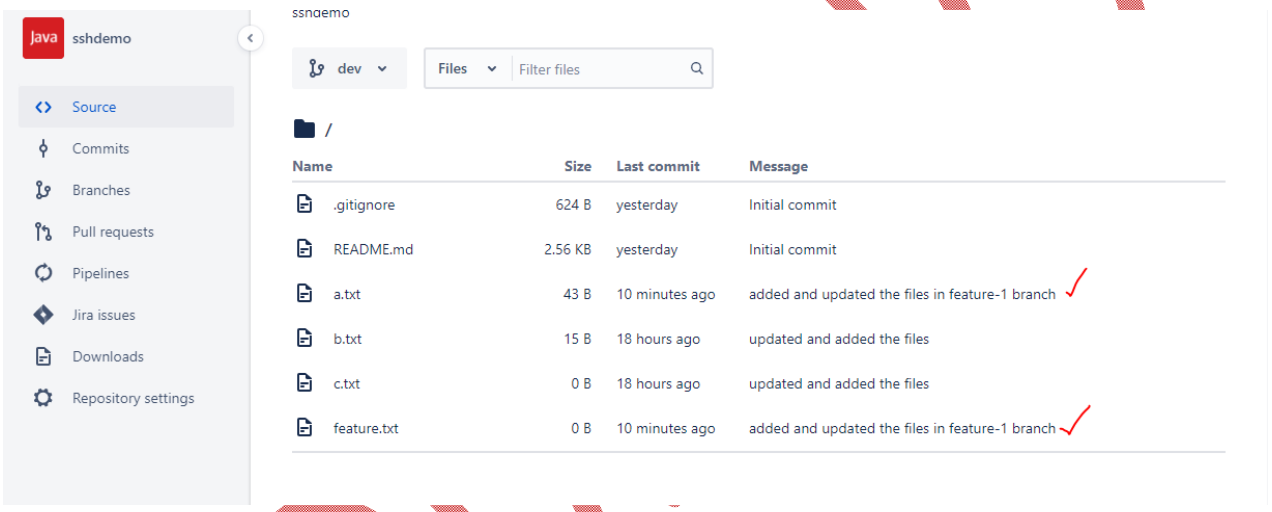




*Prepared by **PPNREDDY** .....*



Verify the *dev* branch



Now *dev* branch having merged code.

README.md file

You can add a README file to your repository to tell other people why your project is useful, what they can do with your project, and how they can use it.

A README is often the first item a visitor will see when visiting your repository. README files typically include information on:

- What the project does
- Why the project is useful
- How users can get started with the project
- Where users can get help with your project
- Who maintains and contributes to the project

Project Title

One Paragraph of project description goes here

Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes. See deployment for notes on how to deploy the project on a live system.

Prerequisites

What things you need to install the software and how to install them and give examples.

Installing

A step by step series of examples that tell you how to get a development env running Say what the steps will be and give the examples. End with an example of getting some data out of the system or using it for a little demo.

Running the tests

Explain how to run the automated tests for this system.

Deployment

Add additional notes about how to deploy this on a live system.

Versioning

Use versioning for each release and keep track of versions.

https://semver.org/

Authors

Add here Authors names.

License

Add the License here.

https://gist.github.com/PurpleBooth/

Git Best Practices

- Use branching strategy and pull requests.
- Commit once you finish the task.
- Avoid merge commits.
- Don't Commit Half-Done Work.
- Test your code before commit.
- Write Good Commit Messages before you are committing and try to use git commands rather than GUI tools.

PSDDEVOPS

*Prepared by **PPNREDDY** .....*