```
==========================
Version Control Softwares
==========================

=> Project contains multiple developers

=> Developers will be working from different locations

Problem-1 : How to integrate all the developers code at once place

Problem-2 : How to monitor/track code changes

                                - who
                                - when
                                - what
                                - why

=> To resolve above problems we will use Version Control softwares

=> We have several version control softwares

                                - SVN (outdated)
                                - GitHub
                                - Bitbucket

=========
Git Hub
=========

=> Git Hub is a cloud platform

=> Using git hub we can maintain source code repositories for the projects

=> Source code repos are used to store project source code at once place

Note: For every project one github repository will be created.

-> Git Repository will provide monitored access (it will track our code changes)

==================
Environment Setup
==================

1) Create account in github

        URL : www.github.com

2) Download and install git client software

        URl : https://git-scm.com/download/win

3) Configure your name and email in git bash.

        $ git config --global user.name "Ashok"

        $ git config --global user.email "ashokitschool@gmail.com"


=================
Git Architecture
=================

1) Working tree
```

```
2) Staging area

3) Local Repository

4) Central Repository


====================
Git Repo practicals
====================

1) Create git repo (public)

        Ex: https://github.com/ashokitschool/inst_mgmt.git

2) Copy script from git repo and execute it in git bash

Note: Git bash is a client to perform git operations with github repo

Note: When we execute git push for first it will ask for git account credentials/token.

Note: It will save our git account credentials in "Windows Credentials Manager". If we want to
connect with diff github account then we have remove those credentials.


====================
Git Bash commands
====================

git init : To initialize working tree

git status : To check working tree status (staged + unstaged)

git add : To add files to staging area

                $ git add <filename>

                $ git add .

git restore : To unstage the files + To discard file changes

                # unstage the file (when it is added to staging)
                $ git restore --staged <file-name>

                # discard file changes (when it is in unstaged)
                $ git restore <file-name>

git commit : Send files from staging area to local repo

                $ git commit -m <msg>

git log : To check commit history

git push : Send files from local repo to central repo.


git clone : To download central repo to working tree

                $ git clone <repo-url>

git pull : To take latest code changes from central repo to our working tree.


git rm : To remove the files
```

```
                        $ git rm <file-name>

Note: When we use 'git rm' command file will be deleted only in working tree. To remove that file
from central repo then we have to commit and push.

                        $ git commit -m <msg>
                        $ git push

============================
How to work with git gui ?
============================

=> Git gui will provide graphical user interface to perform git operations

=> To open git gui we can execute below command in our working tree

                $ git gui

=============
Git Branches
=============

=> Multiple teams will be working on the project paralelly like below

                - on going development
                - bug fixing
                - poc / r&d
                - production support

=> When all these team members merging their code changes then it is very difficult to differentiate
which code changes are done by which team in repository.

******* To overcome this problem we will  use git branches concept *********

=> Git branches are used to maintain seperate code bases for multiple teams working in the same
project.

=> Using git branches multiple teams can work paralelly.

=> In project, one team work shouldn't effect other teams work. We can resolve this issue using git
branches concept.


=> In git repo, we will have branches like below

                        - main (default)
                        - develop
                        - feature
                        - sit
                        - uat
                        - release ....

Note: We can use any name for the branch and we can create any number of branches in git repo.


Note: When we use git clone we will get default branch (i.e main)

$ git clone <repo-url>

# clone git repo develop branch
$ git clone -b <branch-name> <repo-url>

# to display current branch name
$ git branch
```

```
# To switch branch
$ git checkout <branch-name>


============================
What is Pull Request (PR) ?
============================

=> It is used to merge changes from one branch to another branch.


====================================
What is .gitignore file in git repo ?
====================================

=> This file is used to specify which files & folders we don't want to commit to git repo.

        ex : target folder, .project, .settings, .classpath etc...

======================================
How to add collaborator in git repo ?
======================================

=> Go to git repo
=> Go to settings
=> Go to collobarators
=> Add team member email and send invitation

Note: Invited member will get email to become collaborator.

Once inviation got accepted, colloborator can commit to our repo.

=======================
What is git conflict ?
=======================

=> When we are merging central repo changes with local repo then we may get git conflict problem.

=> If two persons working on same file then we may get conflicts problem.

=> When conflict occurs we have to resolve those conflicts and we have to commit without conflicts.

#### Conflicts can occur in 2 scenarios ###

1) When we execute 'git pull' command there is a chance of getting conflicts.

2) When we are merging branches then there is a chance of getting conflicts.


============================
What is git stash command ?
============================

=> It is used to store working tree changes to temp area and make working tree clean.

$ git stash

$ git stash apply

Usecase :
----------

9:00 AM => Task-320 assigned for you
 // your started working on that task
```

```
 // few code changes already done in m1() method
 // you are in middle of the task
```

Note: Task is not completed

12:30 PM => Manager assigned Task-321 and told high pririoty

first complete Task-321 and then work on Task-320

This is task is related to m2 ( ) method.

When we try to commit m2 () changes m1() changes also will be committed because both methods are in the same file.

Bu the requirement we need to commit only m2 () changes to central repo first. Here we can use git stash commnad before starting m2 () method changes.

```
======================
git fetch vs git pull
======================
```

git pull : download latest code changes from central repo and merge with working tree directley. There is a chance of getting conflicts here.

git fetch : download latest code changes from central repo to local repo. It will not merge changes into working tree.

Note: After git fetch completed, we need to execute 'git merge' command to merge changes from local repo to working tree.

```
                git pull = git fetch + git merge
```

```
=============
git revert
=============
```

=> It is used to remove the changes we have commited from remote repository based on the commit id.

```
$ git revert <commit-id>
```

=> After executing git revert command it will open editor then press Esc + :wq to close the editor.

=> After 'git revert' command execution we need to execute 'git commit + git push' to publish revert operation to central repo.

```
====================
what is git fork ?
====================
```

=> forking means creating copy of other git user repository in our git hub account.

```
==================
Realtime workflow
==================
```

1) Management will decide repository server (Ex: github or bitbucket)

2) Management will decide branching strategy (which team should use which branch)

3) Development Team should send request to create git repo to DevOps team with manager approval.

4) DevOps team will manage user permissions on the repository.

5) Once git repo created we can create branches based on our requirement.

6) As a developer we will perform code integration + branch merging with pull requests.

7) When there is production deployment, DevOps team will make code freeze
(remove write permissions on the repository)

Note: If we don't have write permission to git repo then we need to raise request to devops team to get access with your manager approval.

Note: If you have any confusion with git operations/branches/branching-strategy take team members help.

Note: We need to be very careful when we are working with source code repositories because if we do any mistake it will effect other team members work also.

```
======================
Working with BitBucket
======================
```

=> BitBucket software developed by Atlasian company.

=> Bitbucket is used to maintain/manage source code repositories like github

=> For Bitbucket repositories we can use 'git' as client.

```
============================
Q) git merge vs git rebase
============================
```

```
=========
Summary
=========
```

1) What is version control & why ?

2) Git Hub Introduction

3) Git Setup

4) Git Architecture

5) Git Repository creation (public & private)

6) Git bash commands

- git config
- git init
- git status
- git add
- git restore
- git commit
- git push
- git clone
- git pull
- git fetch
- git merge
- git rm
- git log

```
                - git stash
                - git stash apply
                - git branch
                - git checkout
                - git revert
```

7) Working git gui

8) Working with git branches & pull requests

9) Git branching strategy

10) Adding Collobarators

11) Git conflicts

12) Git forking

13) Real-Time workflow

14) BitBucket