K.S SCHOOL OF ENGINEERING & MANAGEMENT, BANGALORE - 109

Department of Artificial Intelligence and Data Science

III Semester

"PROJECT MANAGEMENT WITH GIT"

Subject code: BCS358C

Prepared by: -

Mrs. K. Padma Priya
Assistant Professor
Dept. of AI & DS, KSSEM

Mrs. P S Geetha
Assistant Professor
Dept. of AI & DS, KSSEM

Mr. Naresh
Programmer
Dept. of AI & DS, KSSEM

Course: PROJECT MANAGEMENT WITH GIT			
Type: Ability enhancement course	**Course Code: BCS358C**	**Academic Year: 2023-2024**	
No. of Hours per week			
Theory (Lecture Class)	Practical/Field Work/Allied Activities	Total/Week	Total teaching hours
0	2	2	28 Hours
Marks			
Internal Assessment	Examination	Total	Credits
50	50	100	1

Aim/Objective of the Course:

Students will be familiarized with basic commands of Git and understand how to collaborate and work with Remote Repositories.

Course Learning Outcomes:

After completing the course, the students will be able to

CO1	**Use** the basics commands related to git repository.	**Applying** (K3)
CO2	**Create** and manage the branches	**Applying** (K3)
CO3	**Apply** commands related to Collaboration and Remote Repositories	**Applying** (K3)
CO4	Use the commands related to Git Tags, Releases and advanced git operations	
CO5	**Analyse** and change the git history	

Project Management with Git (BCS358C)

INDEX

Subject : Project management with git **Subject Code : BCS358C**

Version control system

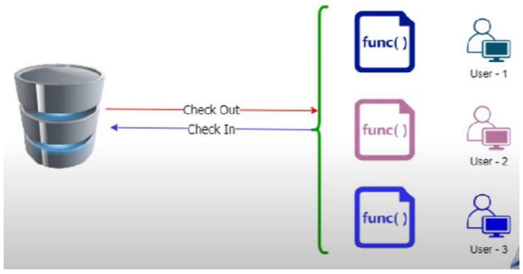
VCS is a software that

- allows you to record changes to your files over time.
- allows for different people to work on the same project at the same time. When multiple users working simultaneously overriding not done
- If you need to revert back to a previous version, you can quickly do so

in version control

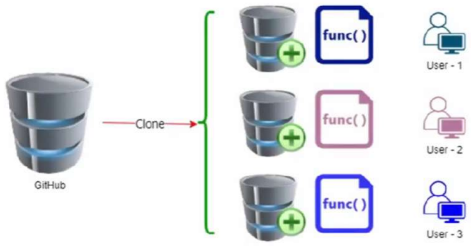
Types of Version Control Systems

Centralized Control systems – in this system files are stored in one system (server) and accessed by the team members. Get latest copy of code and share latest copy of code



Disadvantage: If the remote server goes down, then no one can work on the code. It impacts code development and even result in code loss. The entire project and team come to a standstill during an outage.

Distributed Control Systems – in this every user to have a local copy of the running history on their machine, so if there's an outage, every local copy becomes a backup copy and team members can continue to development offline.



Popular Version control Systems

GitHub, GitLab, Beanstalk, Microsoft Team Foundation Server, AWS CodeCommit, Mercurial

Git

Developed by Linux Community
Born in 2005

Git is most popular distributed version control system (also called as source control system) in the world.

- Free
- Open Source
- Super fast
- Scalable

Difference between git and GitHub –

Git lives on local machine. GitHub is website and runs on server.
Project done in git can be uploaded to github.

How git stores data ?

Snapshot and not the difference

In Difference method

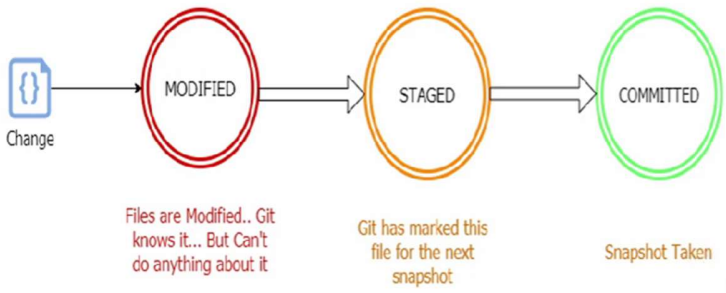
This is a test
This is a new test
Stores only **new**

But in git

This is a test	snapshot - 1
This is a new test	snapshot - 2

Git states

Git file goes through three states



Modified staged and committed

Git commands

Git init – is used to initialize a blank repository in the current working directory
Git commit - is used to save the changes to the local repository. The command helps you keep record of all the changes made.
Git add – is used to add changes in the current directory to the staging area

Git status – is used to display the state of the current repository and the staging area
Git merge - is used to integrate different branches into a single branch

Git push – is used to upload the content from the local repository to the remote repository
Git pull – is used to fetch the new commits and merge them into the local branch
Git clone – is used to create a copy of the target repository or create a clone
in a new directory at a new place.
Git branch – branch refers to an independent line of development. The git branch command is used to create, list, rename and delete branches.
Git check-out – works with branch command to enable navigation between branches

Git config – used to set configurations like the name, email id etc. This information should be provided as soon as git is installed, since it is used by git at every commit.
Git diff – used to show differences between the changes made on a file
Referred to as multi use command that runs the function on different git data sources.
Git log – to view previous commits that have taken places in the git project. When list appears on the screen. It shows the reverse chronological order. Most recent on top
Git reset – to undo the local changes that are made to the state of a git repository. It has three primary forms – --soft, --mixed, --hard
Git rebase – rebasing refers to the moving or combining a sequence of commits. The git rebase command is used to integrate changes from one branch to another.

Experiment No. 1

Setting Up and Basic Commands:

Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.

Solution:

Create a folder learninggit

Open my computer or files in task bar

Right click on window, click on pop up window, click on new folder

Enter name of folder as learninggit

Right click on the folder to open git bash

Configuring git

```
$git config --global user.name geethaps
$git config --global user.email geethapsjc@gmail.com
```

to check if configured

```
$git config --global user.name
$git config --global user.email
```

Creating and Initializing a repository

```
$git init
```

Check the status of files

```
$git status
```

Create one file document1.txt using text editor using vi editor

```
$git status  //shows that document1.txt is untracked
```

Staging the file

```
$git add document1.txt
$git status (shows that document1.txt is staged and to be committed)
```

To unstage the file

```
$git rm --cached document1.txt
```

To stage it back

```
$git add document1.txt
$git add -A (for all files in the folder)
```

Committing the file

```
$git commit -m “My first commit”
$git status (shows nothing to commit)
$git log (shows the commit history with name of commit)
$git log -p -1 (to show only last commit details)
```

To restore if file is deleted (restores till last commit)

```
$git checkout document1.txt
$git checkout -f
```

Make changes to document1.txt by adding a line

```
$git status (shows modified status)
```

Send the modified file to staging area

```
$git add -A
$git status
```

Commit the file

```
$git commit -m “Second commit”
```

or (to commit without sending to staging area)

```
$git commit -a -m “Second commit”
$git status
$git log
```

To commit only one file

```
$git commit -m 'second commit' -- document1.txt
```

Experiment No. 02

Creating and Managing Branches:

Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."

Creating a file document1.txt

$vi document1.txt

Add few sentences, save and exit

Creating a branch

```
$git branch feature-branch
```

Switching to a branch

```
$git checkout feature-branch
$vi document1.txt
```

Add one line in document1.text, save and exit

Commit document1.text in feature-branch

```
$git commit -m “committing in feature-branch”
```

Switch to master branch

```
$git branch master
```

Check if the changes made in feature-branch is visible

```
$cat document1.txt
```

Merge feature-branch into master

```
$git merge feature-branch
```

Once again check the contents of document1.txt for changes.

```
$cat document1.txt
```

Experiment 03

Creating and Managing Branches:

Write the commands to stash your changes, switch branches, and then apply the stashed changes.

Solution:

Create a folder stashDemo

Open terminal

```
$mkdir stashDemo
$cd stashDemo
$git config
$git init
```

Create an empty file and save it

```
$vi index.txt
$git add .
$git commit -m “commit #1 from master branch”
$git branch feature
$git checkout feature
$vi feature.txt
$git add .
$git commit -m “commit #1 from feature branch”
$git switch master
```

Open index.txt and add one line

```
$vi index.txt
$git add .
$git commit .
$git checkout feature
$git log
$vi index file  //add first line
$git status
$git checkout master // error --- commit or stash
                     // error: Your local changes to the following files would be
                     overwritten by checkout: index.txt.Please commit your changes or stash them
                     before you switch branches.Aborting
$git stash           //stash command for performing stashing
      or
$git stash save "stashing first change"
```

```
$git stash list      //command to display the available stashes

$git checkout master        // allowed

$git status                 // work area is clean

$git checkout feature

$git status

$git stash list

$git stash pop          // recently pushed stash is applied and removed from stash

$git stash list
```

Experiment 04

Collaboration and Remote Repositories :

Clone a remote Git repository to your local machine.

Solution:

Create a repository in github webpage, upload some files to it and copy the URL of the repository.

Create a folder learninggit

Open my computer or files in task bar

Right click on window, click on pop up window, click on new folder

Enter name of folder as learninggit

Right click on the folder to open git bash

check the list of files present in the folder

```
$ ls
```

Cloning the repository into the local machine

Copy the URL of a repository from GitHub

```
$ git clone <URL>          // paste the copied URL and the repository will be copied
                           into your local machine
$ ls                       // to see the list of files present in the folder and
                           find the cloned repository present
```

Experiment 05

Collaboration and Remote Repositories :

Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch

Solution:

Create a repository and create two branches and the clone it to your local machine

```
$ git clone <github repository URL>
$ ls                    // the newly added repository can be found
```

Enter the new repository with cd command

```
$ cd <new cloned repository name>        // enter the repository
$ git branch -r                          // displays the branches associated with the
                                         repository
$ git status
                                  //On branch main Your branch is up to date with
                                  'origin/main'.nothing to commit, working tree clean
```

Add a new file in the branches in github and commit the changes and come back to the git bash

```
$ git fetch origin                  // to the see the changes made in the repository
$ ls                                // to see the files from fetch but not added to
                                    the current working directory
$ git log –oneline          //only local commits will be available but not the newly added
                          commits from github
```

Use rebase command to apply local commits on top of the changes fetched from the remote branch

```
$ git rebase origin/login
$ git log --oneline
```

Check the list of files to find the file from remote branch added to the local branch

```
$ ls
```

Experiment 06

Collaboration and Remote Repositories :

Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

Solution:

```
Create a folder mergegit
$mkdir mergegit
$ cd mergegit
$ vi master1.txt          //create two files and save
$ vi master2.txt
$ git add .

$git commit -m "master1 and 2 committed"

$ git branch feature-branch             //create a branch

$ git checkout feature-branch           //switch to feature-branch

$ vi feature1.txt          //create 2 files and save

$ vi feature1.txt

$ git add .

$ git commit -m "feature1 and 2 committed"

$ git checkout master      //switch to master branch

$ ls          //list the files of master branch
```

Merge the branch feature-branch with master with custom merge

```
$ git merge feature-branch -m "merging new to the master"

$ git log –oneline --graph
```

Experiment 07

Git Tags and Releases :

Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository

Soution:

Create a file and save it

```
$ vi gittag.txt
$ git add .
$ git commit -m "gittag committed"
$ git log –oneline  //check the log to know the commit ID
$ git tag v1.0 <any commit ID>
$ git tag           // to see the list of tags
$ git log –oneline          //check the visibility of tag name for the given commit ID
```

Experiment 08

Advanced Git Operations
Write the command to cherry-pick a range of commits from "source-branch" to the current branch.

Solution:

Check the list of files in master branch

$ ls
Create a branch with the name “source-branch”

$ git branch source-branch

Create and save 3 files, add and commit them one by one

$ vi bugfix1.txt
Add some content and save it
$ git add bugfix1.txt

$ git commit -m "bugfix1.txt commiitted"

$ vi bugfix2.txt

Add some content and save it

$ git add bugfix2.txt

$ git commit -m "bugfix2.txt committed"

$ vi bugfix3.txt
Add some content and save it
$ git add bugfix3.txt

$ git commit -m "bigfux3.txt committed"

Check the log for 3 commits

$ git log –oneline

Switch to master to perform cherrypick

$ git checkout master

Perform cherrypick for the commits done on source-branch branch to add them in master branch

$ git cherry-pick <bugfix1 commit ID>..<bugfix3 commit ID>

Check the contents of master branch with the help of ls command

$ ls

The files will be added in the master branch and check the log to see the newly added commits

$ git log --oneline

Experiment 09

Analysing and Changing Git History

Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?

Solution:

Create a file and save, add and commit it

$ vi <filename.txt>

$ git add .

$ git commit -m "committed message"

Create a file and save, add and commit it

$ vi <filename.txt>

$ git add .

$ git commit -m "committed message"

$ git log –oneline

Copy anyone of the commit ID displayed as the output of the got log command

$ git <commit ID> show

Experiment 10

Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."

```
$git log --author="JohnDoe" --since="2023-01-01" --until="2023-12-31"
```

Experiment 11

Write the command to display the last five commits in the repository's history.

Solution:

To display the last five commits in the repositor's history the git log command can be used with -n option:

```
$ git log -n 5
```

If you want a more concise output, you can use the –oneline option:

```
$ git log -n 5 –oneline
```

Experiment 12

Write the command to undo the changes introduced by the commit with the ID "abc123"

Solution:
Check the list of files in master branch

```
$ ls
```

Check the log to identify the commit you want to revert

```
$git log –oneline
```

To undo the changes introduced by a specific commit with the ID "abc 123", you can use the git revert command. The git revert command creates a new commit that undoes the changes made in the previous commit. Here is the command:

```
$git revert <any commit ID>
```

Copy and paste any commit ID or commit reference of the commit you want to undo. After running this command, git text editor for you to provide a commit message for the new revert commit.

Check the list of files in master branch

$ ls

Alternatively, if you want to completely remove a commit and all of its changes from the commit history, you can use the git reset command. Keep in mind that using git reset can rewrite history and should be used with caution, especially if the commit has been pushed to a remote repository.

$git reset --hard abc123.

Again, replace abc123 with the actual commit hash or commit reference. After using git reset hard, you are working directory will be modified to match the specified commit discarding all commits made after it. Be cautious when using --hard as it is forceful operation and can lead to data loss.