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

**Version control system**

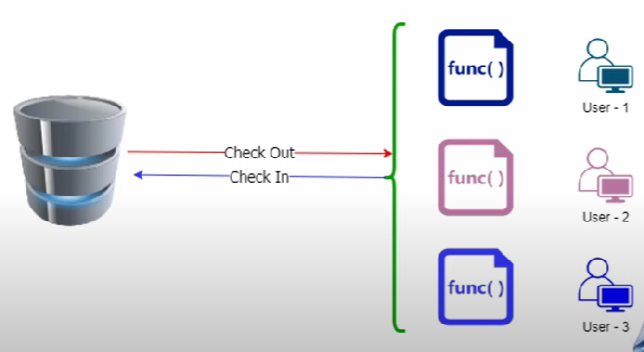
VCS is a software that

* + 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

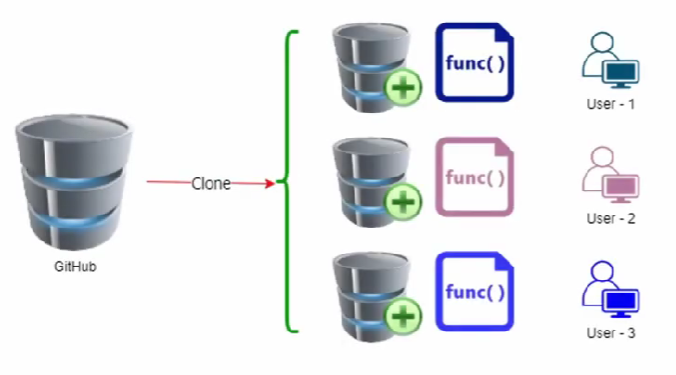
**Types of Version Control Systems**

**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](https://about.gitlab.com/blog/2020/11/19/move-to-distributed-vcs/) 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**

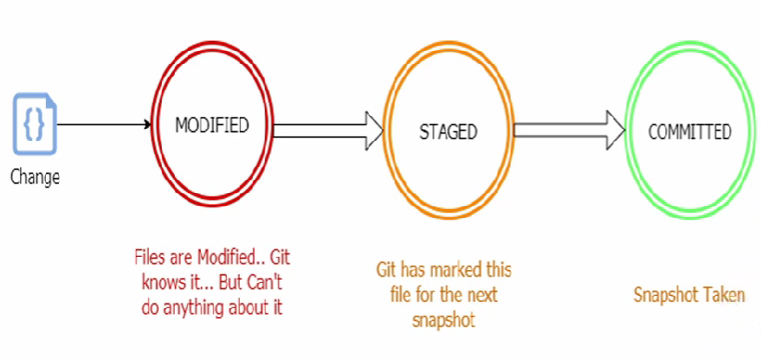
**But in git**

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.

**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](mailto: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.**

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.**

**Experiment 03**

**Creating and Managing Branches**

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

changes.

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

git stash or git stash save "stashing first change"

git stash list

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.

Sol:

Create a repository in github, 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

$ ls // to see the list of files present in the folder

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

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 treeclean

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 branch <new branch name> //create a new local branch

Create a file and commit it from the new branch

$ git checkout <new branch name>

$ vi <new file name>

$ git add . // add the file to the staging area

$ git commit -m "some message" // commit the new file

$ git log –-oneline // to check the commit history

$ git checkout origin //switch to the origin branch

$ git rebase <new branch> // rebase the local branch

to the fetched repository

$ git log –oneline

$ git status // to see that the new branch is added

**Experiment 06**

**Collaboration and Remote Repositories**

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

**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

**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

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 got the visibility of tag name for the given commit ID