EXPERIMENT 7

VERSION CONTROL SYSTEM SETUP AND USAGE USING GIT

AIM

VERSION CONTROL SYSTEM SETUP AND USAGE USING GIT

- Creating a repository
- Checking out a repository
- Adding content to the repository
- Committing data to repository
- Updating local copy
- Comparing different revisions
- Revert
- Conflict and solving a conflict

What is Git?

Git is free and open source version control system, originally created by Linus Torvalds in 2005. Version control systems (VCS) are a category of software tools that help software teams maintain their source code. VCS allows developers to keep track of every modification made to the source code and also to turn back the clock and compare the earlier versions of code to fix their mistakes.

Benefits of using a VCS

- A complete long term history of every files.
- Branching and merging
- Traceability - being able to trace every change made in the software and connect it to project management and bug tracking softwares

Creating a repository

- You can use the UI provided by websites to create a repository and clone the repository to the local computer.
    - to clone an existing repository use *git clone https://example_url.git*
- Use the command *git init* when inside your project's home folder to initialise an empty repository.

Some commonly used git commands

git init	create a new repository
git clone <repo>	clone the repository on to the local machine
git status	get the status of your local repository. It tells you how your project is progressing when compared to the remote repository
git add <filename>	tell git to start tracking the file
git add .	add all files
git commit	commits all the added files. You must provide a commit message in the text editor that opens up
git commit -m”commit message"	a commit command with the message
git push origin master	save the committed changes to server
git pull -all	pull all changes from bitbucket server to your local repository

Branching in Git

Branches are most powerful when you are working on a team. You can work on your part of the project by creating a branch and merging it to the main branch when you have finished. A brach provides an independent area for yourself to work.
There will be a main branch called master by default.

Commonly used commands in branching

git branch new_branch	create a new branch
git checkout new_branch	checkout to the new branch to start using it
git merge new_branch	merge the new_branch to the master branch (perform this operation after checking out to the master branch)
git branch -d branch_name	delete a branch

Forking

You have only read access but not write access to another user’s repository.This is where the concept of forking comes in.
Here is the process of forking a repository

- Fork the repository to copy it to your account.
- Clone the forked repository to your local computer
- Make changes in the repository
- Push the changes to your repository
- Create a PULL REQUEST from the original repository you forked and add the changes you have made
- Wait for the owner of the original repository to accept or reject changes

To fork a repository use the website of the git client you are using.

Undoing changes in a repository

- git checkout

    the git checkout command serves three distinct functions

    - checking out files
    - checking out commits
    - checking out branches

    Checking out a commit makes the entire working directory match that commit.
    This can be used to view an old state of your project without altering the current state in any way.

  How to use checkout

    - *git log --oneline* will show the ID of each commit made
    - use *git checkout <commit_id>* to go to that commit

- git revert

    - git revert <commit> can also be used to undo changes.
    - This generates a new commit that undoes all of the changes introduced in <commit> and apply it to the current branch

- git reset

    - *git reset <file>* is used to remove the file from staging area but leave the working directory unchanged. This unstages a file without overwriting any changes
    - *git reset* is used to reset the staging area to match the most recent commit, but leave the working directory unchanged.

- *git reset* - - *hard* is used to reset the staging area and the working directory to match the most recent commit. The - - *hard* tag tells git to overwrite all changes in the working directory.
- git *reset <commit>* Move the current branch tip backward to <commit> , reset the staging area to match, but leave the working directory alone.
- git *reset* - - *hard <commit>* moves the branch tip backwards to <commit> and resets both the staging area and working directory to match.

Managing conflicts

- When an user rebases or merges conflicts may occur. Conflicts occur when git cannot merge or rebase properly.
- If a merge conflict occurs, we have to resolve the conflict in order to move forward with the merge/rebase
- Run a *git status* to see where the problem is.
- Edit the file to resolve the conflict.
- Add the files again and use *git rebase* - - *continue*
- If you are not able to resolve the conflict, use *git rebase* - - *abort* to abort the rebase. (similarly abort the merge)
- Use git push origin master to push the changes

Comparing different revisions

- git diff <commit> <commit> can be used to compare two different commits
- here the <commit> is the commit id of the specific commit

RESULT

Basic git commands has been familiarised.