Version Control System

**Why use VCS**

* Collaboration
* Storing Versions
* Restoring Previous Versions
* Understanding what happened
* Backup

**Deferent Types of VCS**

* VSS

Microsoft visual SourceSafe is a source control program, oriented toward small software development projects. Now it’s discontinue

* SVN

SVN is the abbreviated form of “Apache Subversion” and is a popular version control system tool it is a centralized version control system

* GIT

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

Another Classification

CVCS – centralized version control system

DVCS – distributed version control system

Installation and Basic Overflow

**Setting up GIT**

Initialize directory 🡪 git init

Check status of files where that are added or not 🡪 git status

Add files to repo 🡪 git add file1.txt file2.txt OR git add .

Commit files with message 🡪 git commit –m “my Message”

Check logs to see commit history 🡪 git log

Befor commit we should run add command

* Git add .
* Git commit –m “updated”

**Commit Past, Present and future**

**Head 🡪** in current branch last commit represent by Head.. e.g. C1 C2 so C2 is head

Previous commit will be count as **head** and future commit will be count as **next.** after commit next will become **head** and **head** will become **past** and a new **next** will be created

**Commit Hash**

Every commit has a unique identifier a 40-character checksum called commit hash

As multiple people can work in parallel, commiting their work offline, without being connected to a shared repository

Therefore, commit hash helps in identifying which user made specific commit

**Unstage Files**

Git reset command will remove file from staging area it will not remove changes only files will come from staging to working area.

Git reset –hard will move files from staging to working area as well as it removes changes to the file.

**Ignoring Files**

In every folder there is .gitignore file where we can specify rules for our project e.g.

Ignore one specific file 🡪 path/to/file.txt

Ignore all files with certain name 🡪 filename.txt

Ignore all files with a certain type 🡪 \*.txt

Ignore all file in a certain folder 🡪 path/to/folder/\*

Branching and Merging

Branches are what need to solve context problem because a branch represents exactly such a context in a project and helps you keep it separate from all other contexts.

When we switch from one to another branch the first branch files will disappear for us.

Show list of branches 🡪 git branch

Show list of branches with some details 🡪 git branch –v

Creates new branch with name ‘new-dev’ 🡪 git branch new-dev

Switch to new-dev branch 🡪 git checkout new-dev

Merge new-dev branch into current active branch 🡪 git merge new-dev

Show commit difference in two branches git log new-dev..master

**Stash**

Commit wraps up changes and saves them permanently in the repository

Stash save changes temporarily.

If you want to switch to another branch without committing changes in current branch, you will need to save you changes somewhere so that you have clean working directory Stash will work in this case.

Later, at any time, you can restore changes from stash in your working copy – and continue working where you left off.

You can create as many stashes as you want.

Save local changes stash clipboard 🡪 git stash

Save local changes in stash clipboard with the name provided in command 🡪 git stash save <name>

Show list of stashes 🡪 git stash list

Apply latest stash and remove it from clipboard 🡪 git stash pop

Apply specific stash and that stash will remain saved in clipboard 🡪 git stash apply stashname

Short-lived branches are about a single topic or feature and once work done with them they are removed

Long-Running branches that are high level and independent of any feature, remain for longer time. They represent states in your project life cycle – like a “production” , “testing”, or development state

Remote Repositories

To clone or download repo from github we copy link and use 🡪 git clone url

After changes to upload 🡪 git push

Push changes to remote repository 🡪 git push

Fetch changes from remote repository 🡪 git fetch

Merge changes that was fetch by ‘git fetch’ command 🡪 git merge

Fetch and merge changes from remote repository 🡪 git pull

Show remote url 🡪 git remote –v

Show details of origin 🡪 git remote show origin

This command will add remote repo in local repository 🡪 git remote add repoName repoURL

If someone already creates repo on their local machine we have to connect and push

For connecting 🡪 git remote add repoName

For push we use 🡪 git push –u repoName BranchName

The –u flag establishes a tracking connection between remote and our local repo.

Git push command require which remote repository you want to push and in which branch of remote repository

Also in git log command you specify remote repository and branch so it will show logs from that branch of remote

**Pull Request**

When working in team we want to review code of team members and merge code in main branch only after that but the problem is without merging we can’t see their code so in this case github provides **pull request** feature.

**Fork**

Fork is server-side to server-side clone means that when you fork someone’s repo it comes to your account.

**Deleting Branch**

For deleting branch 🡪 git branch –d branchName

For deleting remote branch 🡪 git branch –dr dranchName

**Undoing Local changes**

Discard local changes of one file 🡪 git checkout head filename

Discard local changes of all files 🡪 git reset –hard head

**Revert**

To do this 🡪 git revert commitHash

It actually does not delete commit instead undo it.

Git Reset –Hard coomithash 🡪 it move to selected commit and delete next one

Git Reset –keep coomithash 🡪 it move to selected commit and discard next one

e.g. fit reset –hard c3 🡪 file will be moved to c3 and c4 and c5 will be deleted.

**Rebase**

When we merge branches using merge command then there are two possibilities

* Fast Forward

When we make changes to one of two or more branches and then merge this merging is call fast forward. In other word, simple merging is called fast-forward.

* Merge Commit

When we make changes of both branches and need to merge this scenario is called merge commit. There when we merge another additional commit will add which contains difference of both commit.

When we want to represents multiple branches as a single straight line or branch we have to call git rebase instead of git merge.

Rebase Commands

Git rebase BranchName

Details

We have two branches, Branch A and B branch A has three commits and branch B has five commits both have two commits common and similar when we called rebase git will save away temporary that similar commits then add branch B commits to branch A and then temporary commits will add at the end.

**Markdown**

Markdown is a lightweight markup language that you can use to add formatting elements to plaintext text documents

Created by John Gruber in 2004, Markdown is one of the worlds most popular markup languages.

Markdown is often used to format readme files, for writing messages in online discussion forums, and to create rich text messaging using a plain text editor.

Github release its own markdown language based on original markdown called GFM.

Basic writing and formatting sysntex

* Heading
* Styling text
* Quoting text
* Links
* List
* Task list

Heading 🡪 # hello h2 🡪 ## hello