1. **git init**

To initialize empty git repository

1. **git config**

To configure username and email

git config --list –show-origin => to show author

git config –global user.name => to set username globally

git config – global email.user\_email => to set email globally

1. **git status**

To show the current status of project

1. **git help**

To check all the commands available

1. **git help specific\_command**

To get information about specific command

1. **git add file\_name**

To add a file to staging area

1. **git rm - -cached file\_name**

To move file from staging area to working directory

1. **git commit -m “message”**

To save the snapshot in repository

1. **git log**

To check all the commits

1. **git diff**

Shows difference between the files in working area and staging area

1. **git diff –staged**

Shows difference between the files in staging area and repository area

1. **git diff HEAD**

Shows difference between the files in working area and repository area

1. **git mv file1name file2name**

This will let git know that we have renamed the file and not deleted the file.

1. **git restore –staged file\_name**

Moving the file from staging area to working directory

1. **git restore file\_name**

Undo the changes we have done

1. **git checkout branch\_name**

Creating a new branch

1. **git checkout -b branch\_name**

Creating a new branch switching to it.

1. **git branch**

Shows all branches

1. **git branch -m branch\_name**

Renaming a branch

1. **git branch -d branch\_name**

To delete a branch, but it gives warning if there are changes that need to be committed.

1. **git branch -D branch\_name**

To delete a branch forcefully

1. **git merge branch\_name**

Merges change from two branches. Uses fast forward or ORT

1. **git rebase branch\_name**

Updates one branch with another branch by applying the commits of one branch on top of commits of the another branch

1. **git rebase -i branch\_name**

**git rebase -i main => pick which commit to add**

This command provides an interactive rebase that means we can add or remove some commits to add it to another branch. We can add commits of one branch to another using an interactive interface.

1. **git commit – amend**

Modify or change the last or latest commit.

1. **git cherry-pick commit\_id**

Used to apply a particular commit from one branch to another branch. Mainly used if you don’t want to merge the whole branch and you want some of the commits

**\*Git Head =>** Head is a reference to a branch or a commit.

If head points to a specific commit then it goes in a detached head state.

1. **git reset**

To move a branch reset is used optionally copies the data from the repositories to working or staging areas

– hard => moves the files both to working area and staging area

–mixed =>moves the files only to staging area

–soft => does not move the files

1. **git reset commit\_id**

Moves the branch to commit

1. **git reset HEAD –mixed**

Moves files from staging area to working area.

1. **git reset head –hard**

Remove files from the staging area and working area.

‘

\***Git Stash =>** Sometimes you want to switch the branches, but you are working on an incomplete part of your current project. You don’t want to make a commit of half-done work. Git stashing allows you to do so.

The git stash command enables you to switch branches without committing the current branch.

1. **git stash**

To save data without commiting

1. **git stash list**

To list files in stash

1. **git stash pop**

Takes recent stash and puts in working directory

1. **git stash save “message”**

Give the name for the particular stash

1. **git stash apply**

Used to apply the most recent stash to your working directory.

1. **git stash apply stash\_id**
2. **git stash show stash\_id -p**

To check changes in stashed data before pulling

1. **git stash drop stash\_id**

To delete a stash details

1. **git stash branch new\_branch stash\_id**

To create new branch with stashed data

1. **git stash clear**

To clear complete stash details

1. **git checkout -**

Move to the previous branch you were on

1. **git checkout HEAD~2**

Move the head two step behind

1. **git switch branch\_name**

To switch to a particular branch. Can’t move to a particular commit.

1. **git restore**

Remove file from staging to working area. If agin use restore then it will remove file from working directory

1. **git revert commit\_id**

used to create a new commit that undoes the changes made by a specific commit or a range of commits

========================================================================

**GITHUB**

Github allows us to host our git repository in the cloud.

**Git Cloning =>** git clone gets the repository that is not present in your machine based on the url we provide

1. **git clone url**
2. **git remote**

Shows remote repository names

1. **git remote -v**

To view any existing remotes for your repository.

1. **Git remote add <name\_for\_url> <url>**

1. **Git remote rename <old\_name> <new\_name>**
2. **Git remote remove <name>**

Remove remote repository

1. **git push <name\_for\_url> branch\_name**

To add branch to remote repository

1. **git push <name\_for\_url> branch\_name : new\_branch\_name**

Pushing the commit of one branch to another on remote repository

**-u (upstream)** => Allows us to set the upstream of the branch we are pushing. It can be said as a link between the local branch to a branch in the github.

1. **git push -u origin branch\_name**

To set the upstream branch and push at the same time.

Next time it will automatically know which branch to push without specifying the branch name.

1. **git push –set-upstream <name\_for\_url> <branch\_name>**

For setting the upstream branch

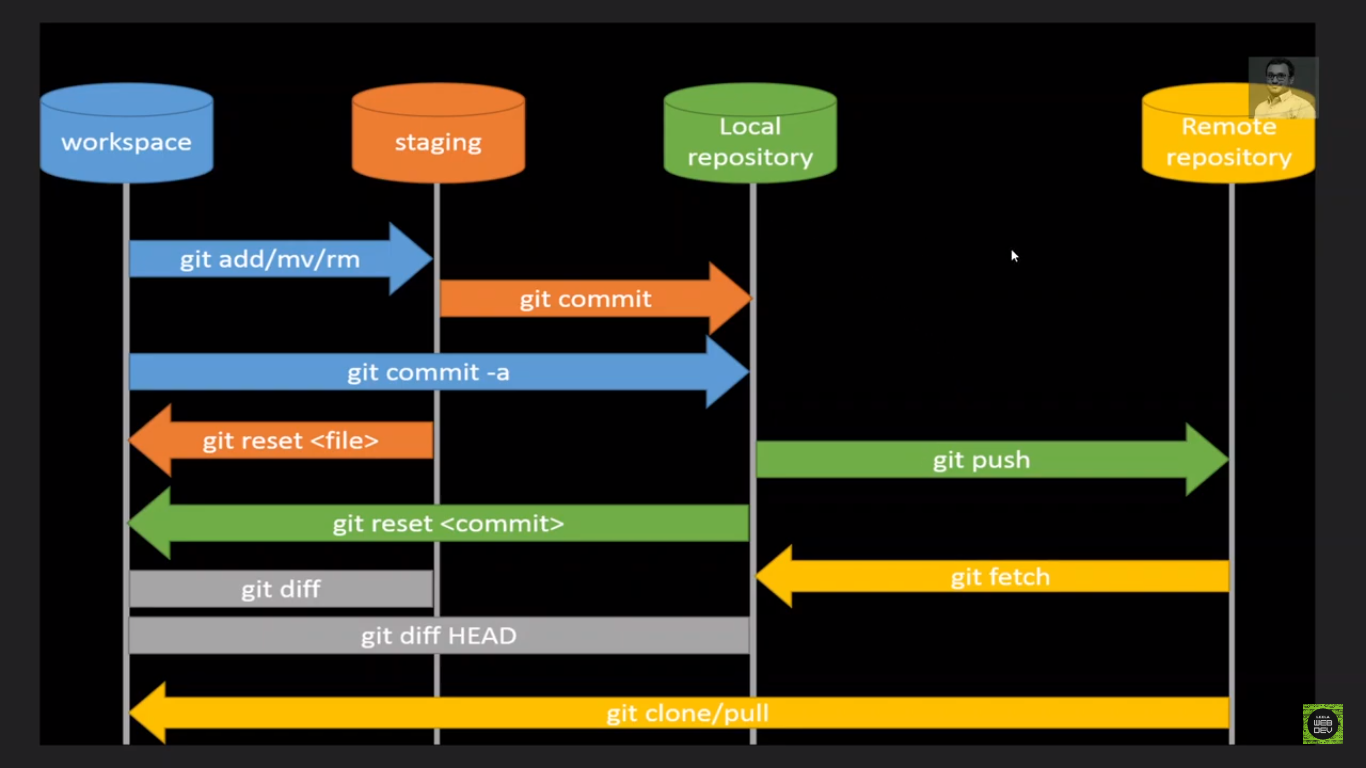
origin/master => remote branch

master => local branch

1. **git branch -r**

To get list of remote branches

Git Fetch => To get the changes from the github repo to local repo.



1. **git fetch <remote>**

Downloads changes from remote repository. But those changes will not be automatically integrated to our working files. It just lets you see what others have been working in, without merging those changes into your local repository.

Go and get the latest info from github, but don’t add to the working directory.

1. **git fetch <remote> <branch>**

This retrieves the latest information from the branch on the remote repository.

1. **git pull <remote> <branch>**

Used to retrieve changes from the remote repository. Unlike fetch, pull actually updates our HEAD branch with whatever changes are retrieved from the remote.

Go and download data from github and immediately update my local repository with those changes.

Git pull = git fetch + git merge

Pull can result in merge conflict.

Sometimes you might have some work locally that is not on github and github has some commits. When you pull down, there may be conflicts.

**Readme** => Used to communicate important information about a repository

**Gists** => Github gists are the simple way to share the code snippets and useful fragments to others.

Every gist is a git repository, which means it can be cloned.

**Github pages** => Public web pages that are hosted and published by GitHub.

**Pull requests** => They allow developers to alert team members to work that needs to be reviewed.

They provide a mechanism to approve or reject the work on a given branch.

Pull request is nothing but merging in the Feature branch.

**Git Forking** => When we are having large open source projects with a lot of contributors.

This forking workflow enables anybody to try and make a contribution for the repo.

There is no permission needed. You can make your own copy. You try making changes and then you make a PR.

When we fork a repo, we basically asking github “Make me my own copy of this repo please”

**Git Tags** => We can tag particular commits so we can label commits by creating a tag, a reference to a moment in time.

Tags are pointers that refer to particular points in Git history.

1. Lightweight tags
2. Annotated tags

**60. git tag**

Print a list of all tags in current repository

**61. git tag -l “wildcard”**

**Git tag -l \*beta\***

Search for tags that match a particular pattern

**62. git checkout tag**

To view the state of a repo at a particular tag. This puts us in a detached HEAD.

**63. git tag <tag\_name>**

Create a tag (lightweight tag)

**64. git tag -a <tag\_name>**

Create a annotated tag

**65. git show <tag\_name>**

To view details of tag

**66. git push <remote\_url> <tag\_name>**

To push tag to remote repository

**67. git push <remote\_url> –tags**

To push all tags to the repository.

**Reflogs** => Reference Logs

**68. git reflog**

**69. git reflog show**

**git reflog show main**

Shows the log of a specific reference(it defaults to HEAD).

**Setting alias** =>

Git config –list –show-origin –global –list

Add this in config file

[alias]

s = status