Git

# For creating directory:

* Mkdir

# For listing the directories:

* Ls

# For showing the folder path of current directory:

* Pwd

# For Telling the Git to Version control a directory:

* git init
* When we use Git init, we have created a local repository
* There is one more way of creating local repository, that is using **git clone**
* Basically, git clone, gets the file from Remote Repository and then creates the local repository

# For Listing the files present in the git local repository:

* git ls-files
* So, those files which has been commited in git are part of GIT LOCAL REPOSITORY
* Those files, which has been just created and NOT been added in GIT LOCAL REPOSITORY are Considered to be PART OF WORKING DIRECTORY only

# For Checking the status of Commits, Changes in the existing file:

* git status

# For adding the file to branch:

* git add <fileName>

# Committing the changes Edited

* Let’s Say, we have edited a file that is already present in git local Repository, now we wanted these changes to be added to git local repository from working directory
* The command for achieving this is
  + git commit -a -m “Commit Message”

# For Commit Changes present in only one file:

* git commit <filename.html> “Commit Message”

# To see the List of all the Commits:

* git log

# For Seeing the Changes in the file after editing:

* git diff <filename.txt>
* Now the above command is for seeing the changes, before the edited changes are added to STAGING from WORKING DIRECTORY
* Now lets say, we wanted to see the changes after ADDING THE EDITED FILE FROM WORKING DIRECTORY TO STAGING then the command is as follows:
* Git diff - - staged <filename.txt>

# For Removing the file from Git Local Repo and Working Directory:

* git rm <filename.txt>
* so once the file is delete, we have UNCOMMITTED changes in the git local repo and working directory
* to Commit the changes as usual, we have to used the command
  + git commit -m “deletion Comments”

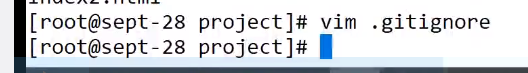
# For Removing the file from Local Repo and Still have the file in Working Directory:

* git rm –cached <filename.txt>

# ignoring few files from git tracking

* Lets say that, there might be a scenario, where we wanted git not to track the changes in certain files or in certain type of files [log files, .class files, email, reports etc].
* Then, we need to create a file called “**.gitignore**” [“<dot>gitignore”] and add the “files” or “type of files”, to that “.gitignore” file
* Once files are added, then commit the “.gitignore” file to git rep.
* After this, whatever “**files**” or “**types of files**” that were added to “.gitignore” file, they will NOT be tracked.

Example:





# For Deleting the Changes made to file present in local Repo

* Lets say that, we have file in local repo and we add some changes to it,
* Now, if these changes are uncommitted, and we wanted to remove these changes or we wanted to UNDO the changes, then the command is as follows
* Git checkout --<filename.txt>

# For Removing the changes to a file from Staging:

* Git reset HEAD <filename.txt>
* After resetting or removing from staging the file will be back to WORKING DIRECTORY only, as UN COMMITTED file.

# Reverting changes to local Repo:

* Lets say that, we add a file and make changes to it and commit the file
* Now this commit will be having a commit id, infact all the commits will be having a commit id
* For seeing the list of commits with commit id, then following is the command
  + Git log –oneline
* Now, after getting the list of commits with commit id, if we want to revert a commit, then following is the command
  + Git revert <commitId>

# RESETTING:

## Hard Reset

* Lets say that, we have been working on a repo for a part one week and we might have done lots of commits
* Now, we might be wanting to DISCARD ALL THE COMMITS PERMANANTLY that has happened for the past one week,
* Then first, we have to get the list of commits till date, using the command
  + Git log –oneline
* Then, we have choose that particular commit id, from which we wanted git to maintain logs as well as files/ commits corresponding to the logs
* Now once we choose the commit id, the following is the command
  + Git reset --hard <commitId>
* Now this is HARD way to DELETE ALL THE COMMITS and CANNOT BE REVERTED

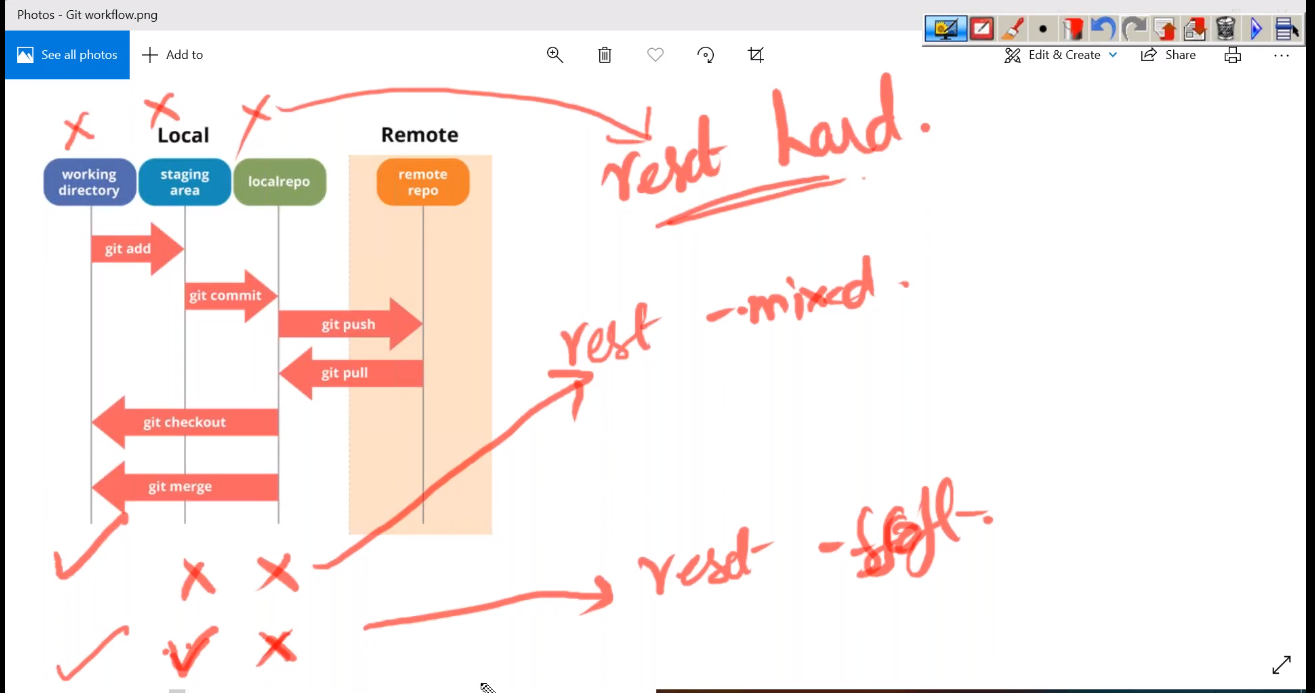
## Soft Reset [Way 1]

* Now, Lets say that, we want to GIT to discard a predefined list of commits with an ability to REVERT or UNDO , then following is the command
  + Git reset --mixed HEAD~2
  + Now, this above command, will be REVERTING or UNDOING the last 2 commits, however the REVERTED CHANGES will be PRESENT IN WORKING DIRECTORY
* NEED TO PRACTICE THIS TO UNDERSTAND MORE

# Soft Reset [Way 2]:

* Git reset --soft HEAD~2
* This command will REVERT or UNDO CHANGES, but will keep the REVERTED CHANGES in Staging

# Summary of Reset:



# For Seeing the List of Branches:

* Git branch

# For Creating a New Branch:

* Git branch <NewBranchName> <FromBranchName>
* Example: git branch b1 master

# For Switching to a particular branch:

* Git checkout <NewBranchName>
* Example: git checkout b1

# For Creating and Switching to a branch in a single command

* Git checkout -b <NewBranchName> <FromBranchName>
* Git checkout -b b1 master

# At Any point of time, if we need to know, the branch that is currently used and active:

* Git branch

# For Merging two branches:

* Git merge <sourceBranch> <destinationBranch>

# Concept of Stashing :

* Lets say that, we are actually, making some changes to the files and we don’t want these changes which might be intermittent to be committed to branch
* However, at the same time, we might not want to discard these changes as well, as we would continue from where we are left off probably the next day
* Now, from GIT perspective, these are changes that are present in working directory, and GIT will continue to show them as Uncommitted changes, But we don’t want GIT to show them as un committed changes as well.
* Just for this kind of scenario, we can the changes to STASH using the following command:
  + Git stash
* Once these changes are added to stash, GIT will no longer show the changes as un committed, but point to remember here is, these changes are still present in Working directory only.
* For listing all STASH’s made so far, following is the command:
  + Git stash list



* Now for seeing the changes made for a particular stash, following is the command



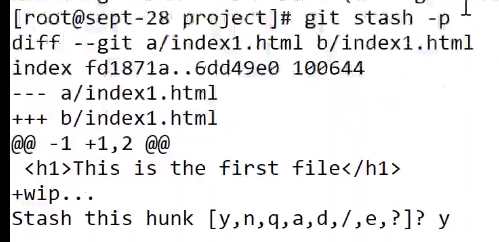
* Now lets say that, we wanted to UNSTASH, for this we have two commands/ ways
  + **Git stash pop <stashId>**
  + example



* + Git stash apply <stashId>
  + Example



* Now, the difference between **apply and pop in stash is**
  + **Apply:** Once the un stashing is successful and the changes made are once again shown by git, the stash which was created earlier, will be still maintained with out deletion
  + **Pop:** Once the un stashing is successful and the changes made are once again shown by git, the stash which was created earlier, will be deleted
* Now lets say that, we wanted to partially stash or stash a particular file, then following is the command
  + Git stash -p
  + Example
  + In the below example, once the command for partial stashing is typed, git will ask, which file to consider, based on the necessity, we have to enable either a “y” or “n” flag



* For deleting a stash:
  + Git stash drop <stashId>
  + Example:



* For Clearing complete all stashes
  + Git stash clear
* If we wanted to stash a NEW FILE that is NOT AT ALL PRESENT in the LOCAL REPO, then the command is
  + Git stash -u

# Rebasing:

* Lets say that, we have master branch with 2 commits whose messages are “Commit 1” & “Commit 2” respectively
* After these 2 commits, lets say that, we create a branch, whose name might be b3, Now this b3 branch also would have above 2 commits of master.
* Now, Lets say that, we have made one more commit in master with message “Commit 3” and then checkout/ switch to branch b3
* After checkout/ switch to branch b3, we make a commit at branch b3, with message “Commit 4”,
* Now, if we merge master with b3, we will see commit message in git log in the following way
  + Commit 1
  + Commit 2
  + Commit 4
  + Commit 3
* Which means the sequence of commits are first from branch b3 and then from master
* Now, if we want, the commit sequence to happen, in the following order, then we need to use the REBASING
  + Commit 1
  + Commit 2
  + Commit 3
  + Commit 4
* The command to use is as follows:
  + Git rebase <sourceBranch>
  + Here, we Lets say that, we are in branch b2 [Active branch], and we wanted to rebase from master branch, then “master” would be SourceBranch
  + Hence the command will be
    - Git rebase master
  + Once the rebase is done, the sequence of commits will be
    - Commit 1
    - Commit 2
    - Commit 3
    - Commit 4

# Pushing Contents to Remote Repo from local Repo

* Command to push
  + Git remote add <keyWordToDefineRemoteURL> <remoteRepoURL>
  + Now, this <keyWordToDefineRemoteURL> would be referred in further git commands, which involves git remote repo
  + Every time, instead of mentioning the <remoteRepoURL>, we can use the <keyWordToDefineRemoteURL>
  + Example:
    - Git remote add origin <https://github.com/devopsUserName/repo1.git>
* Command to push branch
  + Git push <keyWordToDefineRemoteURL> <brancgName>
  + Example:
    - Git push origin master

# For Deleting the Branch

* Git branch -d <branchName>
* For deleting the branch with out any warning
  + Git branch -D <branchName>
* For Deleting the branch on Remote:
  + Git push origin --delete <branchName>

# For Creating a local Repo [Way 2]

* Its done through cloning a remote repo
* Command to clone:
  + Git clone <remoteRepoURL.git>
* When we clone a remote repo, the <keyWordToDefineRemoteURL> is automatically defined by name “origin”

# For updating the local repo with content of remote repo:

* Git pull origin

# To see the difference between remote repo and local repo:

* Git fetch
* Any changes on remote, which were not there locally could be identified using “Fetch” Command

# Thumb Rule:

* Before we push some changes to remote repo, we have to ensure the LOCAL REPO is updated with the contents of REMOTE REPO using the command **git pull origin <branchName>**