1. What is Git?
   1. Decentralized/Distributed
   2. Can be centralized
   3. Created for Linux Kernel Project—contrubuted more than around 12000 devlopers and contains over 50 million lines of code
2. De-facto standard- wide adoption and integration
3. The Repository
   1. Collection of Files Managed by Git
   2. History (all of it)
   3. Working Directory/ Workspace
   4. Quasi-hidden .git Folder
   5. Tip: Focus Purpose of Repository
4. Commits and Files
   1. Snapshots are commits
   2. One or More Files Changes
   3. Dummy File to “save” a Folder
   4. Commits on Timeline (Branch)
5. Git for Windows
   1. Minimal Configuration required
      1. In Git Bash: git config --global user.name “manmath markunde”
      2. In Git Bash: git config --global user.email “[manmath738@gmail.com](mailto:manmath738@gmail.com)”
6. Make Notepad++ as default editor in gitbash
   1. In Git Bash: notepad++ ~/.bash\_profile
      1. inside .bash-profile file add new line as
         1. In Git Bash: alias npp=’notepad++ -multiInst -nosession’ then restard the gitbash
      2. Now set Default text editor in Git us following command
         1. In Git Bash: git config --global core.editor “notepad++ -mulitInst -nosession”
      3. To test Git integration, by using command if the notepad++ opens git default configuration file then configuration successfull.
         1. In Git Bash: git config --global -e
7. P4Merge for Windows ([http://perforce.com](http://perforce.com/)& option- HELIX P4MERGE: VISUAL MERGE TOOL
   1. Visual Compare and merge Tool
   2. Command line Merging is annoying
   3. this is an Optional- But Highly recommended
      * 1. In Git Bash: git config --global diff.tool p4merge
        2. In Git Bash: git config --global difftool.p4merge.path “[C:/Program](../../C:/Program) Files/Perforce/p4merge.exe”
        3. **Troublishooting: Try replacing** [**C:/**](../../C:/) **with /*c*/ instead**
        4. In Git Bash: git config --global difftool.prompt false
        5. P4Merge as graphical Merge Tool
           1. In Git Bash: git config --global merge.tool p4merge
           2. In Git Bash: git config --global mergetool.p4merge.path “[C:/Program](../../C:/Program) Files/Perforce/p4merge.exe”
        6. In Git Bash: git config --global mergetool.prompt false
        7. git version
8. Auto create demo folder and within this folder git generates .git folder

In Git Bash: git init demo

* + - 1. alternate, using bash go to the demo directory and use command to generate .git folder

In Git Bash: git init . ["." represents current directory]

1. README.md
   * + 1. The ".md" extension is for markdown, which is a popular file format.
2. Git Staging
   * + 1. staging is used to prepare for next commit

In Git Bash: git add "fileName"

* + - 1. this will add untracked working directory into git staging area, to verify that the file is in staging area by using command

In Git Bash: git status

* + - 1. Now,the file will show status as - unstage and changes to be commited, for that use command

In Git Bash: git commit -m "Commiting Message"

* + - 1. after that if we check again git status command then it will show that, repository on branch master and nothing to commit, working directory clean
      2. by mistake user just used command "git commit" then a default text editor will open and user has to put commit message on that file

1. View all commits log and show

In Git Bash: git log

* + - 1. command will show all the commit messages of current repository/branch In Git Bash: git show
      2. command will show the last commit and a diff containing all the changes.

1. command will show only the tracking files which are being track by git.

In Git Bash: git ls-files

1. The express command
   1. which will first add modified file into staging and then commits message, the express command can't work for untracked files
      1. In Git Bash: git commit –am “commit message”

-a : stands for adding modified files into git staging area

m: after then commiting with commit message

1. Backing out Chages (undo)
   1. backing out can be work on only unstaging. Use following command to unstage a file
      1. In Git Bash: git reset HEAD “file name”
   2. after unstaging the file use another command to checkout or discard the changes in working directory
      1. In Git Bash: git checkout -- “file name”
   3. NOTE: if user commits any modification then this steps will not work
2. In Git Bash: git help log
3. Below command is for log details
   1. git log --oneline will show history of all commit’s in the single line instead of multiple
   2. git log --graph will show asterik (\*) based graph denoting branching level hierarchy
   3. git log --decorate will show which part of which branches
   4. git log --all will show all branch history
   5. In Git Bash: git log --oneline --graph --decorate --all
4. Creating git alias is an shorten existance of longer command
   1. In Git Bash: git config --global alias.hist “log --oneline --graph --decorate --all”
5. Once this done. To use this command
   1. In Git Bash: git hist here “hist” is an alias name of the command
   2. In Git Bash: git hist -- “File name” : only information related to file
6. View all alias list
   1. In Git Bash: git config --global --list
7. Rename & deletion of file
   1. In Git Bash: git mv “old File” “new FileName”
   2. In Git Bash: git rm “filename”

After performing above command this works in stage mode then commit with message and changes will reflect on Working Directory.

1. Managing Files outside git
   1. In Git Bash: git add -u add update file to staging mode
   2. In Git Bash: git add -A covers all modification possibly and make updates accordingly git index
2. Ignore/Excluding Unwanted Files
   1. Check where “.gitignore” file is available or note in current directory, if not then make one. Syntax for this file is one pattern or expression per line.
   2. In current directory, user want to exclude the any “.log” file, then in “.gitignore” mention “\*.log” then all the extention “.log” will not be shown in git staging area.
      1. In Git Bash: git add .gitignore AND commit this with message.
3. Comparing Differences using tool
   1. To add p4merge as visual merge tool,
      1. In Git Bash: git config --global diff.tool p4merge
      2. In Git Bash: git config --global difftool.p4merge.path 'C:\Program Files\Perforce\p4merge.exe'
   2. To setup p4merge as visual diff tool
      1. In Git Bash: git config --global merge.tool p4merge
      2. In Git Bash: git config --global mergetool.p4merge.path 'C:\Program Files\Perforce\p4merge.exe'
4. Comparing Differences
   1. HEAD: points to the last commit of current branch [ HEAD called as pointer]
   2. To check all commits from stating to the end in the current branch.
      1. In Git Bash: git diff “commit Id” HEAD
   3. Same by using p4merge tool, but this will iterate up to the all the last modified file which are in unstage area.
      1. In Git Bash: git difftool “commit Id” HEAD
5. Branch
   1. To check current branch , In Git Bash: git branch
   2. Follows command will create new branch “updates” and also switched to newly created branch. If previous branch modifications are pending to commit in master branch then this command also carry the modification to new branch.
      1. In Git Bash: git checkout -b “branch Name”
   3. In Git Bash: git diff updates master here “updates” is a branch name.
   4. In order to change/ switch to the branch
      1. In Git Bash: git checkout master
   5. Merging Branches: Fast forward merging done by following command, that first go to the master branch
      1. In Git Bash: git merge “branch Name”
      2. Now if we check git hist then we will see that HEAD is pointing to the both merge branches
   6. TO delete the branch, In Git Bash: git branch –d “branch Name”

Note: The history never goes away, it is just the label changed.

1. Branch Merging with conflict problem: for that we create a scenario, where crate another branch and inside that modify a file, the modification should be done in same line with master branch. After that use merge command then it will show the error about conflict. For that use external difftool GUI to merge by manual and save the file
2. Git Tags- git is an ocean of tags

List of Tags In Git Bash: git tag – list

Delete Tag In Git Bash: git tag -d “Tag Name”

View associated information of tag In Git Bash: git show “tag Name”

* 1. Light weight
     1. In Git Bash: git tag “tag Name”
  2. Annotated tags which include extra information associated with tags
     1. In Git Bash: git tag -a “tag Name” -m “commit message”

1. Git Stash – save working directory for temporary changes
   1. In Git Bash: git stash this will temporary makes changes to use any other changes in different file, or do other operations.
   2. Then come back to the previous stash command to make unstaging the working directory In Git Bash: git stash pop
2. Time Travel with reset and reflog
   1. --soft – least destructive, basically points as HEAD
   2. --mixed
   3. --hard – More destructive than others

[**https://github.com**](https://github.com/) **for Remote Repository**

1. Go to the URL login to github then create a new repository with name you like.

Now to link **github** repository with our local **git** repository or push an existing repository from the command line / using Git Bash

* 1. In Git Bash: git remote -v command tells that any remote connection with github
  2. Please make sure that HTTPS link you copied from github
     1. In Git Bash: git remote add origin “github repository link”
     2. add subcommand takes two parameters,first the name of remote reference we want to create -”origin” (name can be any but primarily it should be origin)and second fill URL to the remote repository
     3. Once the command runs successfully check wheather you see two links in Git Bash for pushing and fetching.

In Git Bash: git remote -v

* 1. In Order to synchronise our local repository with remote repository use push command
     1. In Git Bash: git push -u origin master --tags
     2. -u which going to setup tracking branch relationship between master branch on local repository and master branch in remote repository.
     3. origin : Name of the remote repository
     4. master : Name of the branch that we are going to push on remote
     5. --tags : send all the tags that are available on local repository to the remote repository
     6. for future push or another push -u is no longer required

1. **SSH Key Generation (Ubuntu)**
   1. First create directory with name .ssh and enter in it.
      1. In Git Bash: ssh-keygen -t rsa -C “Email address”
         1. -t : type whichis rsa
         2. -C : common Name
      2. then prompt will ask where to save the keys, press enter for default location as “/home/ghost-rider/.ssh/”
      3. Next, will prompt for password, again for password confirmation
      4. From this directory list the fils you will see two keys with name
         1. id\_rsa
         2. id\_rsa.pub
      5. Leave first id\_rsa and open id\_rsa.pub with your editor and copy the content/key
   2. Now go to the GitHub.com, Then go to Settings, Then ssh, Now paste content/key.
   3. Now got to git bash login ssh
      1. In Git Bash: ssh -T [git@github.com](mailto:git@github.com)
      2. It will prompt to Enter ssh password that you created
   4. then you will See “Hi Ghost-Riders! You've successfully authenticated, but GitHub does not provide shell access.”
   5. Now repeat step 30 with SSH link. Hence using ssh you will no longer prompt for userid and password.
2. Clone Repository
   1. In order to create a local copy of clone copy HTTPS/SSH any URL from GitHub.com of your repository (Link option in your repository will be CLONE or DOWNLOAD) In Git Bash: